NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SERVICE-ORIENTED ACCESS CONTROL

by

Joseph W. Lukefahr

September 2014

Thesis Advisor:	Dennis Volpano
Second Reader:	Geoffrey Xie

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE 09-26-2014	3. REPORT TYPE AND DATES COVERED Master's Thesis 07-01-2012 to 09-26-2014

4. TITLE AND SUBTITLE

SERVICE-ORIENTED ACCESS CONTROL

5. FUNDING NUMBERS

6. AUTHOR(S)

Joseph W. Lukefahr

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

As networks grow in complexity and data breaches become more costly, network administrators need better tools to help design networks that provide service-level availability while restricting unauthorized access. Current research, specifically in declarative network management, has sought to address this problem but fails to bridge the gap between service-level requirements and low-level configuration directives. We introduce service-oriented access control, an approach that frames the problem in terms of maintaining service-level paths between users and applications. We show its use in several scenarios involving tactical networks typically seen in the military's field artillery community.

14. SUBJECT TERMS

computer network, network management, network design, data communications, network security, network access control, network service

15. NUMBER OF PAGES 99

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU

THIS PAGE INTENTIONALLY LEFT BLANK

SERVICE-ORIENTED ACCESS CONTROL

Joseph W. Lukefahr
Captain, United States Marine Corps
B.S., Texas A&M University, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
from the

NAVAL POSTGRADUATE SCHOOL
September 2014

Author: Joseph W. Lukefahr

Approved by: Dennis Volpano
Thesis Advisor

Geoffrey Xie
Second Reader

Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

As networks grow in complexity and data breaches become more costly, network administrators need better tools to help design networks that provide service-level availability while restricting unauthorized access. Current research, specifically in declarative network management, has sought to address this problem but fails to bridge the gap between service-level requirements and low-level configuration directives. We introduce service-oriented access control, an approach that frames the problem in terms of maintaining service-level paths between users and applications. We show its use in several scenarios involving tactical networks typically seen in the military's field artillery community.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ACL	access control list
BGP	Border Gateway Protocol
DHCP	Dynamic Host Configuration Protocol
DNS-SD	DNS-based Service Discovery
DOD	Department of Defense
FML	Flow-based Management Language
HTTP	Hypertext Transfer Protocol
HTTPS	Secure Hypertext Transfer Protocol
IDS	intrusion detection system
IEEE	Institute for Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPv4LL	Internet Protocol Version 4 Link-Local Addressing
IRC	Internet Relay Chat
LAN	local area network
MAC	media access control
mDNS	Multicast DNS
NAT	network address translation
OSPF	Open Shortest Path First
PDU	protocol data unit

QoS	quality of service
RIP	Routing Information Protocol
SOAC	service-oriented access control
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
USMC	United States Marine Corps
VLAN	virtual local area network
VPLS	Virtual Private LAN Service
VPN	virtual private network
WLAN	wireless local area network
Zeroconf	zero-configuration networking

Acknowledgments

First and foremost, I thank God for his grace in giving me the ability and motivation to complete this thesis. It is only by that grace that I could have accomplished such a feat. As a result of this research effort and my time at the Naval Postrgaduate School, I have learned that I can surely do all things through Jesus Christ who strengthens me.

I thank Dennis Volpano for his outstanding expertise, patience, and mentorship as my thesis advisor for the past year. He allowed me the freedom to explore new ideas, yet he knew when and how to guide me in my work. He knew just how to encourage and motivate me to tackle challenging problems. He once said, "Cracking hard nuts can be frustrating." I will remember this the next time I face an overwhelming problem. Most importantly, he was legitimately concerned about my professional development and personal well-being. I also thank Geoffrey Xie for his help not only with my thesis but also as an instructor of several advanced networking courses. Through his knowledge and teaching style, I gained critical knowledge and skills that will certainly help me in my upcoming assignment.

I thank my classmates for their help and optimism over the past two years. Iron sharpens iron, and because of the high-caliber individuals who sat in class with me through some difficult exams and excruciating lectures, I have become a better student, researcher, thinker, and Marine. I look forward to working with each of them again back in the fleet.

Finally, I cannot thank my wife enough for her patience and perseverance. While I was working late nights, early mornings, and holiday weekends, she was raising two young children. Words cannot express my gratitude to my family for their support. I love them so much.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Computer network complexity continues to grow rapidly. Bridges, routers, firewalls, and other network devices flood the marketplace. Each new device provides some new capability in allowing the network administrator to manage his network. That new capability brings the added requirement of configuring the device to ensure proper operation without compromising security. Furthermore, proper operation requires proper care and maintenance. A network administrator who is competent and properly trained, therefore, becomes even more necessary for meeting these added demands.

However, as networks change in size and complexity with the introduction of each new technology, the task of ensuring service-level availability and access control becomes more and more daunting. The tools that today's network administrator must use to accomplish this task are inadequate. At this point in time, the network administrator must rely on industry best practices, experience, and trial-and-error methods to design and secure his network.

The discrepancy between the increasing complexity of today's networks and the network administrator's ability to manage that ability continues to grow. As it grows, the security of the world's sensitive information becomes more and more vulnerable to theft and exploitation. This, in turn, leads to more and more incidents of cyber attack and data loss, costing companies, government organizations, and individuals millions of dollars.

One only needs to read any daily news source to find reports of these incidents. Stories of disgruntled students hacking into school servers to change grades are common. Reports of data loss by small businesses are seen frequently. Loss of sensitive information by large corporations like Target [1] and TJX [2] are becoming more and more common in international news. Government organizations like the Department of Veterans Affairs are experiencing more and more attempts to exfiltrate sensitive information from their servers.

Many of these incidents find their causes in networks that are poorly designed or access control mechanisms that fail to achieve the organization's access control policy. Network administrators, despite their best efforts using the tools at their disposal, cannot plug every security hole and secure every service. As networks and their complexity continue to grow, best practices and trial-and-error methods cannot meet the increased security requirements, and the result is costly.

Network administrators need proper tools to manage this complexity. They need ways to systematically design networks and implement access controls. They should be able to specify service-level access control policies and then automatically and systematically derive a network configuration that necessarily and sufficiently satisfies those service-level requirements. Our research seeks to address this issue. This introductory chapter provides some motivation for this issue, defines this problem in more specific terms, and sets the stage for our work.

1.1 Problem Statement

Ultimately, we seek a solution to the problem that underlies many of the data breaches and network attacks mentioned earlier. In order to achieve that solution, we must first state the problem in concrete terms. This section presents that problem statement and some terminology used throughout this work.

We specifically describe the aspects of this problem in later chapters, but we provide brief definitions here. A *physical topology* is a description of the physical arrangement of devices in a network and the physical connections between those devices. A *service specification* is simply a representation of the services or resources available on a network, as defined by their various standards documents. Finally, a *logical organization* is a description of how devices are configured to control the flow of data within the network (e.g. virtual local area network (VLAN) configuration, access control list (ACL) placement, or routing design). Changing the network's logical organization affects how data flows in a network and therefore affects users' access to services.

With these definitions, we state our problem, the service-oriented access control problem, as follows. Given a network's physical topology, a specification of a service that is to run on the network, and a policy governing which users have access to the service, output a logical organization of the topology, if it exists, that achieves the policy.

1.2 Research Description

The main thrust of this research is to design a framework within which the service-oriented access control (SOAC) problem can be solved for a limited set of hardware and a limited set of possible services. For a given network and service specification, multiple logical organizations may exist. It should be understood, then, that the ultimate goal of this research effort, but beyond the scope of this thesis, is to develop an algorithm to compute the optimum solution based on some criteria. In this work, our focus is a framework and methodology that will allow us to find a solution, if one exists.

Furthermore, the range of possible devices, services, and configuration options is extensive, so our work focuses on a limited set of devices with limited capabilities. Specifically, we focus on Internet Protocol (IP) networks with routers that utilize static routing and stateless packet filtering, Ethernet switches with media access control (MAC) filtering capabilities, and typical hosts and servers. Likewise, the range of network services in existence is vast, so we limit our focus to a subset of these services in order to convey the basic concepts of the framework. Specifically, we focus on Hypertext Transfer Protocol (HTTP), Secure Hypertext Transfer Protocol (HTTPS), and Internet Relay Chat (IRC). We also constrain our work in terms of network size. As computer networks can grow quite large, we focus our work on small-scale tactical networks like those found in the military's battalion-sized units.

Notice, however, that the SOAC problem does not have any specific logical organization as input. Any algorithm for the problem must produce such an organization and is free to explore all possibilities. Therefore, a solution to a SOAC instance may be correct according to its service policy but involve network configuration that does not comply with industry best practice or human intuition.

Our work is guided by several research questions. These questions, whose answers are discussed throughout this work, are listed below.

1. Can a network's logical organization be automatically derived based on its physical topology and service requirements?
2. What constitutes a logical organization?
3. What constitutes a network service?

4. What constitutes a service specification?
5. How can a network's logical organization change based on the service specification?

1.3 Research Benefits

This study benefits all organizations that rely on computer networks to communicate and share information. Each of these organizations has a network of physically connected devices and a set of services that must be supported by that network. Therefore, each of these organizations finds itself dealing with an instance of the SOAC problem.

This study benefits the Department of Defense (DOD) in particular because the data and services on its networks are particularly sensitive. The national security of the nation heavily depends of the security of the networks maintained by the DOD. Therefore, the need for tools that aid in secure network construction is critical. This work helps meet that need and helps ensure the security of the nation's secrets.

1.4 Organization

Chapter 2 provides background information on this research topic. Several sections describe the current state of research in this area and how previous work contributes to our research. This chapter also provides motivation for our approach to the problem.

Chapter 3 details the model that we use to frame the problem. It introduces abstractions for the network's physical topology, abstractions for the corresponding logical organization, and an operational semantics to describe network behavior.

Chapter 4 describes the design of our access control logic used to reason about denying access to network resources.

Chapter 5 shows how we can use the SOAC approach to configure a common tactical network. Three scenarios show the application of the framework on tactical networks seen in United States Marine Corps (USMC) field artillery battalions.

Chapter 6 includes our conclusion of the research and recommendations for future work.

CHAPTER 2:
Related Work

Efforts have been made to ease the burden of network management. What these efforts have in common is a way to specify a particular logical organization in a high-level language that is then reasoned about for properties like conflicts, reachability, black holes, etc. and ultimately translated automatically into low-level device configuration commands. Contrast this with SOAC. It does not include any logical organization as input. A solution to SOAC *is* a logical organization. For this reason, SOAC is similar to the problem of zero-configuration networking (Zeroconf) except the latter does not include a service description as input. Zeroconf merely tries to establish local-area connectivity and a name space.

We contrast earlier efforts in network management and Zeroconf with SOAC in more detail in the following sections. In Section 2.1, we survey the fields of network management and network design, relating various works to SOAC. In Section 2.2, we introduce Zeroconf and explain the fundamental assumption that separates Zeroconf and SOAC. In Section 2.3, we investigate static reachability analysis and its relation to our work. As we will see, SOAC is a problem that is much different from the problems addressed by declarative network management and zero-configuration networking.

2.1 Network Management and Design

The field of current network management research that most closely relates to our work is declarative network management. This research aims to ease the task of network management by allowing the network administrator to specify network policy in a declarative manner, typically using relational logic. With a declarative network management approach, the network administrator expresses what properties must be present in the network. Various mechanisms and engines take that policy and convert it into lower-level configuration directives to implement that policy in the network.

Narain *et al.* [3] present a declarative network management system based on model finding. Motivated by the large gap between end-to-end infrastructure requirements and detailed implementations, they offer a solution for four fundamental problems in network management: network requirement specification, configuration synthesis, configuration error diagnosis, and configuration error repair. They express configuration information as a database containing variables and configuration requirements as constraints on those variables. These requirements typically describe desired subnetting or IP address allocation. The key to the system, therefore, is a requirement solver that takes a database and a set of requirements as inputs and tries to compute values for the variables that satisfy the requirements. The requirement solver uses a model finder to convert requirements expressed as first-order logic into Boolean constraints that serve as inputs for a SAT solver. The SAT solver provides either a solution or a proof of unsolvability.

This work also addresses several issues relevant to our discussion. The motivation for the work of Narain *et al.* [3] lies in the large gap between high-level infrastructure requirements and the low-level configuration directives that meet those requirements. We notice a trend in the design of these network management systems. Some collection of variables and values make up the configuration space. Some knowledge base of network behavior is captured in a systematic manner. Some set of requirements or desired outcomes, specified in terms of the knowledge base, is provided. Finally, some engine instantiates or constrains the variables based on the knowledge base and desired outcomes. However, the system described by Narain *et al.* contains a knowledge base that only expresses knowledge of the network. For example, they describe several requirements about IP addresses in the network, including the requirement for unique IP addresses for hosts on the same subnet. It does not justify these requirements in terms of granting or denying access to services. Furthermore, we cannot tell how failure to meet the requirement affects the service. An instance of SOAC includes a service specification and establishes a relationship between the network and its services, hence the service-oriented nature of the problem.

Hinrichs *et al.* [4] provide an approach to network management using a declarative language to specify policy and specialized network hardware to implement that policy in the network. Motivated again by the inadequacy of traditional network configuration techniques, the authors present Flow-based Management Language (FML), a flow-based declarative network policy language. This language allows a network administrator to specify a rule containing several characteristics of a network flow – including source and target users, source and target hosts, and source and target access points – and whether that flow should be allowed, denied, or relegated to a certain path in the network. A security policy, therefore, is a list of rules enforced in the network. The authors provide several examples of policies for various uses, including access control, network address translation (NAT), and quality of service (QoS). The authors implement this language within NOX [5], an operating system for OpenFlow [6] controllers, and deploy policies specified in FML on networks consisting of a series of OpenFlow switches.

This work provides some interesting insight and some potential overlap with a SOAC approach. The authors [4] describe flows in terms of users, hosts, access points, and protocols. FML ties flows to the network in two ways. First, for each flow, the policy writer specifies source and destination access points. In the policy statements $allow(U_s, H_s, A_s, U_t, H_t, A_t, Prot, Req)$ and $deny(U_s, H_s, A_s, U_t, H_t, A_t, Prot, Req)$, A_s and A_t are network access points. Specifying these access points effectively ties flows to network elements besides source and destination hosts. Second, the policy writer specifies network nodes through which flows are specifically routed (with the *waypoint* keyword) or through which flows are specifically not routed (with the *avoid* keyword). SOAC requires that we focus simply on users and services. We should consider the network a “black box”, where we are not necessarily concerned with the details of the network configuration as long as it is correct with respect to the service-level requirements. We are not necessarily concerned with routing of flows through a network as long as the service-level goals are met.

Chen *et al.* [7] present a database-driven declarative system for network management and operation. This system’s design is motivated by the increasingly complex task of network management, by the difficulty of network-wide reasoning, and by the dynamic nature of networks. The authors’ key observation about the state of affairs in network management is that a systematic expression of domain knowledge (i.e., relationships between various network protocols and between protocols and the network devices that support them). Their system, therefore, captures this knowledge and exposes high-level primitives to simplify and automate both device-specific and network-wide management while minimizing the need for human intervention. The building blocks of this system include a data model that stores network device configuration and status, a set of rules representing the previously mentioned domain knowledge, and an engine that leverages the data and ruleset to provide database-like operations like data queries, insertions, and deletions. Rules, as described in the paper, typically describe how some high-level network behavior depends on lower-level network or device configuration. The resulting capabilities, therefore, are network-wide reasoning, automatic configuration (given the ability to write configuration data), and network policy enforcement.

This work seems to address many of the issues that we pose in our introductory discussion. Chen at. al. [7] share our motivation that networks are growing in complexity and that tools available to network administrators today fail to manage that complexity. They also identify the lack of a systematic description of network behavior as a major factor in the discrepancy. They also seek to provide high-level primitives and abstractions to simplify the task of describing network properties. For example, they simplify the task of configuring a Virtual Private LAN Service (VPLS) connection, which is difficult when performed manually, with the single database query `ActiveVPLSConnection.insert(int1_id,int2_id)`. This approach focuses on providing high-level network abstractions for network administrators to use in realizing network policy. Though these higher-level abstractions are useful, the focus here is the expression of network policy. In SOAC, we need an approach that focuses on users and their access to services. Network organization and relationships between network elements should not be taken as input but rather generated as a part of a solution to a SOAC instance.

Several other works [8]–[11] in the field fit within the same basic framework. One of the common themes in this research is that a network administrator specifies a network policy, usually expressed in terms of network elements or relationships between network elements. For example, FML requires a policy writer to define a flow by specifying users, hosts, access points, protocol, and whether only requests should be considered. In order to specify the access points, the writer must have some understanding of how access points and hosts relate to each other in the network. When considering the SOAC problem, these approaches seem to fall short because they tie high-level policy to the network. A SOAC instance requires a policy governing which users have access to which services, without reference to underlying network elements.

With this understanding, the fundamental difference between declarative network management and SOAC lies in the inputs of the two problems. A SOAC instance does not include a policy regarding flows, like FML, or relationships between network elements, like Narain *et al.* Instead, the only sort of policy input to a SOAC instance is a service access policy governing which users may or may not access a given service. A declarative network management approach entails expressing a policy that the network must be configured a certain way, algorithmically deriving that configuration, and then using experience and industry best practices to justify why that policy achieves the required access control. This gap between network policy and service-level requirements motivates our work. We seek an approach that is more rigorous and complete than relying on experience and industry best practices.

Departing from declarative network management, Sung *et al.* [12] focus on network design in enterprise networks. They share our motivation of simplifying the task of network management, and they specifically look at the unique challenges of designing enterprise networks (i.e. the need for highly customized designs with wider ranges of security, resilience, and performance requirements). They decompose the task of enterprise network design into four steps: (1) physical topology planning, (2) VLAN and link-layer design, (3) routing design, and (4) reachability control. In this work, they address VLAN design and reachability control and propose systems for accomplishing each. For VLAN design, a system takes as input a mapping of hosts to organizational groups and a table of traffic patterns, in kilobits per second, between groups. It outputs VLAN assignments for hosts and placement of designated router and root bridge for each VLAN, all such that traffic cost is minimized. Sung *et al.* define this cost as the sum of broadcast traffic, inter-VLAN data traffic, and intra-VLAN data traffic. For reachability control, a system takes as input a reachability matrix between VLANs and a set of topology-changing events to which the network must respond while meeting all requirements in the reachability matrix. It outputs recommended ACL placement based on both correctness and feasibility criteria. Multiple solutions for ACL placement may meet correctness and feasibility criteria, so the designer chooses one of four placement strategies based on preference. Sung *et al.* evaluate this approach by applying it to a large-scale campus network topology and comparing broadcast traffic, data traffic, and ACL correctness between the currently running network and the systematic design. They also used systematic ACL placement to discover an inconsistency in access control on the currently running network.

Concerning SOAC and this thesis, this paper [12] provides several points of discussion. First, the reachability matrix, originally proposed in previous work [13] and applied here, closely resembles the notion of a service policy that a SOAC instance takes as input. In the reachability matrix M_R, traffic permitted from VLAN i to VLAN j is denoted by $M_R(i,j)$ and is expressed as a set of packets whose header information meet certain criteria given in first-order logic. Expressing reachability in terms of traffic between VLANs does not match our intent to express a service policy only in terms of users and their access to services. However, we could adapt the reachability matrix to capture application-layer messages allowed (or not allowed) between hosts, which better suits our purposes. Second, Sung *et al.* propose four strategies used as preferences for ACL placement that is certainly useful in SOAC. They observe that multiple solutions for ACL placement may exist and that some strategy for choosing the best solution should be employed. In the same way, multiple solutions may exist for a SOAC instance, so some strategy for choosing the best solution should be employed. Third, the size of networks addressed by Sung *et al.* is much larger than those addressed in this thesis. Whereas we consider small-scale networks like those supporting battalion-sized units, they focus on large-scale enterprise networks consisting of thousands of hosts and hundreds of routers and switches. Despite the difference in scale, this work is still relevant specifically in its discussion of systematic reachability control and its heuristics for ACL placement, as mentioned earlier. However, it is constrained to achieving access control through VLAN assignment and ACL placement. Sung *et al.* show how this approach mirrors the most common practice in enterprise network design. Nonetheless, we seek an approach that allows us to utilize the entire logical organization space to determine a solution. A solution to a SOAC instance may involve VLAN assignment and ACL placement, but such tasks are not necessary. An equally correct solution may not involve either of these practices.

2.2 Zero-Configuration Networking

The field of Zeroconf takes a unique approach to taking the guesswork and human intervention out of network design and administration. The uniqueness of Zeroconf lies in its ultimate goal. The initial requirements document [14] describes this goal, where network hosts automate the task of network configuration. In home networks, mobile networks, ad hoc networks, and other emerging networks, there may not be adequate personnel or knowledge available to administer the network properly. Therefore, Zeroconf seeks automatic configuration of networks without manual intervention or central administration. Though originally conceived as a solution in smaller ad hoc networks, some research efforts seek to extend this functionality to larger networks, including enterprise networks.

Zeroconf is a collection of complementary network protocols that, when employed on a network of devices with connectivity at the data link layer, accomplish four specific tasks required to enable networked applications to communicate over an IP network. Those four tasks, as specified in a host profile [15], are listed below.

1. IP interface configuration
2. Translation between host name and IP address
3. IP multicast address allocation
4. Service discovery

A Zeroconf implementation is essentially a suite of protocols that collectively accomplish these tasks. For example, popular Zeroconf implementation, Apple Bonjour [16], implements Internet Protocol Version 4 Link-Local Addressing (IPv4LL) [17] for interface configuration, Multicast DNS (mDNS) [18] for host name resolution, and DNS-based Service Discovery (DNS-SD) [19] for service discovery. In this protocol, a service provider advertises a service that is accessible to any other host that understands the protocol. Bonjour does not specifically address multicast address allocation. These three protocols work together to achieve the goal of automatic network configuration for supporting services and networked applications.

Early Zeroconf implementations accommodated small single-router ad hoc networks (e.g., a home local area network (LAN)). Some research efforts seek to extend this functionality to larger multi-router networks. Single-router Zeroconf solutions do not address some issues associated with multi-router networks. Specifically, single-router solutions fail to provide dynamic exchange of routing information between routers and consistent assignment of IP subnets in the network. These issues motivate recent work in the field. Akinlar *et al.* [20], [21] present algorithms to address these issues in multi-router networks. With these issues addressed, Zeroconf implementations can ensure IP connectivity between all hosts in a multi-router network and can therefore facilitate operation of the higher-level host name resolution and service discovery protocols throughout the network. Therefore, they can enable automatic configuration and service discovery in larger networks like tactical and small organizational networks.

Fundamentally, Zeroconf addresses the need to enable communication without manual intervention or central administration. Because of the lack of central administration, therefore, individual hosts are responsible for controlling access to their services. However, the only access control mechanism available to hosts is not advertising services. This lack of fine-grained security control is simply unacceptable in most use cases, even in home networks where certain family members like children may not access certain services. SOAC assumes that a central administration authority is available to provide a service specification. Therefore, a SOAC approach enables that authority to implement a fine-grained access control scheme for the network's resources that Zeroconf simply cannot provide.

Furthermore, a major difference between a Zeroconf approach and SOAC lies in what network properties are configured. As previously stated, a specific protocol focuses on unique IP address and subnet assignment among devices. Once it achieves this, host name resolution and service discovery protocols allow service-level communication. The only network configuration that occurs in this process is assignment of IP addresses. Though this may be enough to enforce a security policy in some cases, we see the need to leverage as many configuration options as possible in order to achieve the fine-grained security needs of tactical and organizational network administrators.

2.3 Static Reachability Analysis

Static reachability analysis is about analyzing a description of a network's logical organization to determine whether the organization admits packets between given hosts. Xie *et al.* [13] introduce the notion of static analysis for determining reachability across a network, and they outline several advantages of the approach. First, static analysis allows a network administrator to compute all possible packets that could travel from a given source to a given destination. Second, static analysis allows a network administrator to compute all possible paths that a given packet could take and, therefore, all possible destinations that a given packet at a given location could reach. Third, static analysis mitigates risk by allowing analysis before network deployment and before a problem arises. Fourth, a network designer can use static analysis to determine whether a network's implementation (i.e. low-level configuration) meets the designer's intent.

Xie *et al.* [13] describe how to compute reachability in a network using static analysis of router configuration files. They model a network as a graph $G = (V, E, \mathcal{F})$, where V is the set of routers, E is the set of directed edges defining connectivity between routers, and $\mathcal{F}$ is a function that annotates edges in E such that, for each edge $\langle u, v \rangle \in E$, $F_{u,v} \in \mathcal{F}$ is the set of packets that the network can carry from u to v. $F_{u,v}$ is also expressed as a packet filter $f_{u,v}$ containing predicates that test properties of packet p, returning true if $p \in F_{u,v}$. This enables reasoning about packet flow over multiple routers by simply applying set operations and first-order logic. For example, determining the set of packets allowed from router u through router v to router w is done taking the intersection $F_{u,v} \cap F_{v,w}$ or by determining the conjunction $f_{u,v} \wedge f_{v,w}$. Furthermore, forwarding state (i.e. collective contents of each router's forwarding table) influences reachability, so they define $F_{u,v}(s)$ as a function mapping the network's forwarding state s to the set of packets allowed from u to v when the network is in state s. With these primitives, Xie *et al.* show how to compute network-wide reachability bounds using classical graph algorithms.

More recent work in this area extends static reachability analysis in various ways. Feamster and Balakrishnan [22] use static analysis to detect Border Gateway Protocol (BGP) configuration faults. Mai *et al.* [23] cast static analysis Kazemian *et al.* [24] introduce a protocol-agnostic framework that statically analyzes network device configurations to identify reachability failures, forwarding loops, traffic isolation, and leakage problems.

In terms of inputs and outputs, however, these works address a common problem. This problem takes as input some representation of a network's physical topology and logical organization, usually in the form of router configuration files. It outputs information about useful network-wide properties, like reachability or configuration consistency. From this perspective, static reachability analysis fundamentally differs from SOAC. Whereas static analysis takes configuration as input and provides network-wide properties as output, SOAC requires network-wide requirements, in the form of users and their access to services, as input and provides configuration as output.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Network Model

A TCP/IP network is a complex system. It consists of devices that communicate with each other through interfaces using a set of complex communications protocols. It is possible, however, to reduce this complex set of protocols and vast range of configuration parameters into concepts that are both simple to understand and relevant to a specific set of network properties. For reasoning about network access control (i.e., developing a formal logical method for ensuring controlled access to network resources), much of this complexity becomes unnecessary. Therefore, an important step toward achieving the formal method is developing a model that eliminates as much complexity as possible without losing information needed to reason about the properties in a real network. For example, it would be unsatisfactory to model network performance without considering link-layer protocols for resolving contention if multiple users share a network link.

In this work, we sought to build a rudimentary framework for approaching the issues associated with service-oriented access control. Therefore, we focused on developing a model that eliminates unnecessary complexity and simplifies the task of reasoning about service oriented access control. We also focused on capturing the basic operation of the network rather than one that tackles all possible features and capabilities used to achieve an access control strategy. However, we kept a high level of granularity that enables reasoning not only about a datagram's movement between devices in the network but also about its movement between layers of the network stack.

To this end, we developed a model employing several abstractions that we describe in this chapter. Section 3.1 describes the network's physical topology and how it is represented. Section 3.2 describes the network's logical organization, how it is represented, and how it relates to the physical topology. Finally, Section 3.3 describes abstractions used to model the network's behavior, specifically abstractions to represent datagrams and to describe how they are transported across the network.

3.1 Physical Topology

A network's physical topology is one of the inputs to this problem. Therefore, it is important that we capture all necessary information about a network's physical topology in a manner that facilitates reasoning about access control within the network. To this end, we designed a structured mapping to organize and label this information to support our approach.

We organize this information into a mapping of devices and interfaces to their unique attributes. For example, the mapping D represents a network and provides information about the network's devices, interfaces, and connectivity. Each device in D is uniquely named, so for a device m on the network, $(D\,m)$ provides information and attributes unique to m. This allows us to reference attributes unique to m within D.

Though a physical topology can be extremely complex and detailed, we are only concerned with the necessary pieces of information needed for our purposes. These necessary pieces of information are described in this section. We begin by introducing the notion of a device's type. We then show how a device's network interfaces are named and organized. We then show how we capture direct connectivity between interfaces. Finally, we describe how we capture hardware-level addresses.

3.1.1 Device Type

If D represents a network, and m names a device in the network, then $(D\,m).type$ is a set of labels that indicate the type of m. The device's type corresponds with its basic functionality and capabilities. For example, an IP router is generally capable of receiving an IP packet at one interface and forwarding it to another interface on the device for transmission based on the longest matching network prefix of the packet's destination IP address [25], [26]. Therefore, if the device m is of type $router$, or if $router \in (D\,m).type$, then m has this IP routing capability within the model.

In this work, we considered clients (which we called hosts), IP routers, servers, Ethernet switches. Therefore, $(D\,a).type \subseteq \{host, router, server, switch\}$ for every device $a \in domain(D)$. IP routers have the packet forwarding capability described above as well as stateless ingress packet filtering and egress packet filtering on all interfaces. Servers generally listen for incoming requests and have similar ingress and egress filtering capabilities. Ethernet switches [27] can forward Ethernet frames and have ingress and egress filtering capabilities based on MAC address. A device may have more than one type. For example, a device may act as a server providing one service while acting as a client requesting another service.

3.1.2 Network Interfaces

Each device connects to and communicates with the network through at least one network interface. In many cases, devices communicate through several interfaces. Ethernet switches, for example, may contain 24 or more network interfaces. Each interface contains some unique attributes that must be captured.

For a device m, $(D\,m).ifaces$ is a set of the names of all interfaces on the device. Each interface is uniquely named, so if interface u is on device m, then $u \in (D\,m).ifaces$. Furthermore, an interface can be uniquely referenced in order to preserve the interface's unique attributes. Information about u can be referenced by using the notation $(D\,m).ifaces[u]$.

3.1.3 Direct Connections

For the interface u on m, $(D\,m).ifaces[u].direct$ provides information about its directly connected interfaces. This attribute is a set of names of all interfaces that have direct physical-layer connectivity with u. By direct connectivity, we mean that connections are established such that link-layer communication is possible. In an Ethernet network, for example, $(D\,m).ifaces[u].direct = \{v\}$ if u is connected to interface v on device n via an Ethernet cable and if u can send an Ethernet frame to v. However, v is not necessarily able to send a frame to u; we would have to say that $(D\,n).ifaces[v].direct = \{u\}$ in order to capture such a symmetric connection.

Because of (1) the ability to capture a one-to-many relationship between connected interfaces and (2) the ability to capture asymmetric physical connections, this model can represent wireless networks as well as wired networks. Though we do not explore wireless networks specifically in this work, the model could be easily extended to allow for this. If we have a wireless local area network (WLAN) [28] interface w that can communicate with interfaces x, y, and z, then $(D\,q).ifaces[w].direct = \{x,y,z\}$.

3.1.4 MAC Address

For every network interface on every interface, except those interfaces belonging to an Ethernet switch, we require a hardware-level address, or MAC address, that uniquely identifies that interface on the network segment. $(D\,m).ifaces[u].hwaddr$ provides this address for interface u. We consider this attribute to be part of the physical topology of the network, and therefore part of our problem's input, because an interface's MAC address is most commonly set within the interface's firmware.

We do not specify a format for this attribute. Though we only considered the ubiquitous 48-bit MAC address format specified in most Institute for Electrical and Electronics Engineers (IEEE) 802 standards, the model remains format-independent. Throughout this work, we represent 48-bit MAC addresses as six bytes, each written as two hexadecimal digits, separated by colons. For example, $(D\,m).ifaces[u].hwaddr =$ 45:09:ED:3A:01:BB.

3.2 Logical Organization

We capture information about the network's logical organization in the same way we capture information about its physical topology. In addition to physical characteristics like MAC addresses and direct connections, the mapping of devices and interfaces to their unique attributes also includes attributes configurable by network administrators. We organize the attributes of a network's logical organization just as we organize attributes of its physical topology.

In a network represented by the mapping D, $(D\,m).ifaces[u]$ provides logical information like IP address for the interface u on device m just like it provides physical information like MAC address. In this section, we describe how we capture pertinent information about a network's logical organization, specifically how we represent IP addresses and subnet masks, forwarding table entries, and packet filter information.

In this thesis, logical organization is assumed not to be a function of dynamic events like link or device failure. Logical organization affects access control, and an enterprise network running spanning tree algorithms would preserve access control in response to link failures. However, there are cases where organization is a function of failure. An example is a route that changes based on an interior or exterior gateway routing protocol. An algorithm for

SOAC assumes it has the final say in all routing, and therefore if there are any physical changes to network topology then the algorithm must be re-run to get a new organization.

3.2.1 IP Address and Subnet Mask

$(Dm).ifaces[u].ipaddr$ provides the network-level address, or IP address, for the interface u on device m in network D. We consider the IP address to be part of the network's logical organization because, unlike the MAC address, the IP address of an interface is most commonly set by the device's operating system (or by a user with administrative rights on the device).

Likewise, $(Dm).ifaces[u].netmask$ provides the interface's subnet mask. This attribute is closely related to the IP address, is also commonly set within the device's operating system settings, and is therefore included as a part of the network's logical organization.

Because of its ubiquity at the time of this work, we use the addressing scheme specified by IP version 4 [29] in this model, and we use $ipaddr$ to label this attribute. In this work, we represent IP addresses in dotted decimal form (i.e., as four bytes) with each byte written as a decimal number from zero to 255, separated by dots. However, it would be possible to adapt this model to accommodate other layer 3 logical addressing schemes. Since the IP address and subnet mask, therefore, are ultimately strings of 32 bits, common bitwise operations can be performed on them. In this work, we use the bitwise AND operation (represented as the $\&$ symbol), the bitwise OR operation (represented by the $|$ symbol), and the bitwise negation or one's complement (represented by the $\neg$ symbol) as operations on these attributes.

3.2.2 Forwarding Table Entries

IP routers generally contain some sort of table structure containing a mapping of networks to interfaces. They use this table to properly route packets to their destination. Therefore, the information in this table is an important part of a network's logical organization. Because this table relates networks to interfaces, we capture the information in this forwarding table in an attribute at each interface. For an interface u on device m, $(Dm).ifaces[u].dest$ is an address governing which packets will be forwarded out interface u upon arriving at other interfaces of m. This attribute follows the same format as the IP address and the subnet mask. The bitwise operations mentioned earlier also apply to this attribute.

3.2.3 Ingress/Egress Filters

Packet filters are an integral part of a network's logical organization and access control policy. In this work, we consider the use of stateless ingress and egress packet filtering as part of an access control policy. This sort of filtering ability is common in most modern operating systems.

If u names an interface on device m in network D, then $(D\,m).ifaces[u].ifilter$ is a function mapping a header field to a set of values used in filtering incoming packets. Likewise, $(D\,m).ifaces[u].efilter$ is a function mapping a header field to a set of values used in filtering outgoing packets. For example, $(D\,m).ifaces[u].ifilter[dstip]$ is a set containing all IP addresses that, if matched to a packet's destination IP address, will cause that packet to be discarded. More formally, a packet p is filtered at interface u if $p.dstip \in (D\,m).ifaces[u].ifilter[dstip]$. We may also write $p.dstip \notin (D\,m).ifaces[u].ifilter[dstip]$ to explicitly state that p will not be filtered. In addition to $dstip$, we use the field $srcip$ for source IP address filtering.

We also use $srcport$ and $dstport$ for filtering on transport-layer segments. In these filter sets, however, we also include the transport-layer protocol in the set to differentiate between popular transport-layer protocols like Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). For example, we block transport-layer segments destined for TCP port 80 with the statement $(tcp, 80) \in (D\,m).ifaces[u].ifilter[dstport]$.

In addition to packet filters at the network and transport layer, filtering at the link layer based on MAC address is also a common ability. Not only can we expect most modern operating systems to be able to filter based on MAC address information, but we can also expect switches to filter based on this information. Therefore, we use the sets $srchw$ and $dsthw$ for this MAC filtering. Similar to packet filtering, we block a frame f based on source MAC address by stating that $f.srchw \in (D\,m).ifaces[u].ifilter[srchw]$.

3.2.4 Transport-Layer Port Range

$(D\,m).portrange$ specifies the set of valid transport-layer port numbers on a device m in network D. Based on TCP [30] and UDP [31] specifications, port numbers are 16-bit fields that each correspond with 65,536 possible ports on an interface. Therefore, $(D\,m).portrange$ typically consists of these 65,536 different port numbers. Though this set is not typically part of a network's logical organization, we use it in the network semantics, which is described later, to capture a client's ability to assign a random valid port number to an outgoing transport-layer segment.

3.3 Network Behavior

With network access control, we are concerned with controlling the flow of messages throughout the network. If we want to control access to a particular server, then we should focus on controlling traffic to and from that server. In order to do that, however, our model must allow us to reason the logical organization influences that traffic. Furthermore, the model must be simple enough to allow us to focus on the task of reasoning about access control.

In general, our model achieves this by casting network behavior in terms of translating, or rewriting, values between variables. A value corresponds with a datagram, also known as a protocol data unit (PDU). The variables that can hold these values correspond with network interfaces. These values are rewritten between variables according to the network semantics. This section describes this model of network behavior in detail by describing the datagram abstractions used as values, the network interface abstractions that take the form of variables, and the network semantics that describes how translations of values between these variables can occur.

3.3.1 Protocol Data Unit Values

Protocol data units (e.g., Ethernet frames, IP packets, and TCP segments) are represented as values that can be assigned to variables and rewritten between variables. A value is a string that serves as a simple representation of a PDU. It consists of information contained in a datagram that is pertinent to access control. The following pseudo-grammar specifies the values used in this thesis:

$$A ::= \mathsf{HTTP\_RQ} \mid \mathsf{HTTP\_RS} \mid \mathsf{HTTPS\_RQ} \mid \mathsf{HTTPS\_RS} \mid \mathsf{IRC\_PRIVMSG}$$

$$S ::= (protocol, srcport, dstport, A)$$

$$P ::= (srcip, dstip, S) \mid (srcip, dstip, \mathsf{ICMP})$$

$$F ::= (srchw, dsthw, P) \mid (srchw, dsthw, \mathsf{ARP})$$

The set A contains all application-layer messages. Though this set can be extensive, it is confined to the values described above to help illustrate the basic concepts proposed in this work. The set S contains all transport-layer segments, so the value s can represent a segment, where $s \in S$. Likewise, P contains all network-layer packets, so $p \in P$ represents a packet. Finally, F contains all link-layer frames, so $f \in F$ refers to a single frame. Header information is represented using dot notation. For example, an IP packet $p \in P$ has a header with a source IP address $p.srcip = 10.0.0.1$ and a destination IP address $p.dstip = 10.0.0.100$. Encapsulated data is represented in a similar way. The packet p encapsulates a TCP segment $s \in S$, so $p.data = s$. Likewise, an Ethernet frame f encapsulates p such that $f.data = p$.

3.3.2 Interface Variables

Our model is built on assumptions that the network consists of several devices, each with one of more network interfaces, and that each of those interfaces is directly connected to one or more distinct interfaces via the physical layer. With this assumption, we can say that each interface in the network is uniquely named and has two variables associated with it: an ingress variable and an egress variable. A network interface v has an associated ingress variable v_i and an associated egress variable v_e. Each variable can hold a value as previously described.

Therefore, we can specify a notation for showing rewrites of values between variables using the rewrite operation. Consider an example where we wish to show a rewrite of a packet p from the egress variable associated with interface u to the ingress variable associated with interface v. We formally represent this with $(p, u_e) \rightarrow (p, v_i)$. Likewise, we show encapsulation of a packet p within a frame f by the rewrite $(p, u_e) \rightarrow (f, u_e)$.

3.3.3 Network Semantics

We use a system of inference rules to codify commonly accepted network behavior (as specified by Internet Engineering Task Force (IETF) requests for comment, IEEE standards, and other standards documents) as translations of values between variables. A standards-based semantics for an IP network over Ethernet with the simplicity and granularity required for our framework is given in a whitepaper [32] and reproduced in Appendix A.

By exploiting the transitive properties of the semantics, the system allows us to take service-level availability requirements and determine constraints on the network to realize those requirements and prove their availability. For example, the following rule captures requirements for frame transmission from an arbitrary device m to a non-switch device n:

$$\frac{\begin{array}{l} switch \notin (D\,n).type \\ f \in F \\ y \in (D\,m).ifaces[w].direct \\ m \neq n \\ f.dsthw = (D\,n).ifaces[y].hwaddr \\ f.srchw \notin (D\,n).ifaces[y].ifilter[srchw] \\ f.dsthw \notin (D\,n).ifaces[y].ifilter[dsthw] \end{array}}{D \vdash (f, w_e) \rightarrow (f, y_i)}$$

The transmission is represented as a translation of frame f at variable w_e to variable y_i, where w is an interface of m and y an interface of n. In order for this to occur, y must be directly connected to w via the physical layer, the destination hardware address of f must match the hardware address of y, and the frame must not be blocked by an ingress filter at y. Those requirements are expressed in the antecedents of the rule above the horizontal line. Below the line is the conclusion that, with respect to the network D, frame f at egress variable w_e can be rewritten as a frame at ingress variable y_i provided the conditions in the antecedent of the rule are satisfied.

Note that this rule is actually a rule scheme because $ifilter$ is unspecified. A concrete definition of $ifilter$ produces a concrete semantics. Say, for example, we have $(D\,n).ifaces[y].ifilter$ instantiated such that the following is true:

$$\text{45:09:ED:3A:01:BB} \in (D\,n).ifaces[y].ifilter[srchw]$$

Therefore, the concrete semantics would prescribe that rewrites are impossible between these two interfaces for frames having this source hardware address.

To further explain the how the network semantics captures knowledge of network behavior, we contrast the previously mentioned rule, named (FRAME-TX-RX), with a rule that captures requirements for frame transmission from an arbitrary device m to a switch labeled n. This rule, named (FRAME-TX-SWITCH-RX), is listed below:

$$\frac{\begin{array}{l} switch \in (D\,n).type \\ f \in F \\ y \in (D\,m).ifaces[w].direct \\ m \neq n \\ f.srchw \notin (D\,n).ifaces[y].ifilter[srchw] \\ f.dsthw \notin (D\,n).ifaces[y].ifilter[dsthw] \end{array}}{D \vdash (f, w_e) \rightarrow (f, y_i)}$$

Notice here that (1) the stated conclusion is the same in each rule, (2) both rules require that w directly connect to y, and (3) f not be filtered at y. Also notice that the (FRAME-TX-RX) rule requires that n not be a switch and $f.dsthw = (D\,n).ifaces[y].hwaddr$. In contrast, the (FRAME-TX-SWITCH-RX) rule states that this conclusion can also be derived if n is a switch, without any need to check the destination address.

The network semantics given in Appendix A focuses exclusively on device protocol handling. Links are not represented in any way. As SOAC is about controlling access to services and not concerned with link bandwidth, information about links, other than their existence, is absent. Also absent in the semantics is the handling of any protocol that could influence logical organization (e.g. Routing Information Protocol (RIP), Open Shortest Path First (OSPF), or Dynamic Host Configuration Protocol (DHCP)). An algorithm for SOAC must have control over all such organization and allowing these kinds of protocols to run in the network would undermine it.

CHAPTER 4:
Access Control Logic

A network semantics like the one described in Chapter 3 and listed in Appendix A codifies commonly accepted network behavior in a system of inference rules. This system allows us to infer the presence application-layer connectivity given a network's topology. Each conclusion reached in the semantics, which speaks in terms of rewrites allowed between variables, represents the presence of some communications path between an application running on a host and an application or service provided by a server. This is useful in automatically generating constraints on a network configuration for the purpose of allowing access to services on a network. However, the semantics does not address the issue of denying access to services on a network.

We can develop a similar system that allows us to infer the absence of application-layer connectivity as well. This system, which we call the access control logic, allows us to reason about denying rewrites between variables such that each conclusion reached in the logic represents the absence of all possible communications paths between an application on a host and one on a server. Derivations in the semantics are existential in nature, where a derivation in the semantics corresponds with allowing a single path through the network between source and destination. In contrast, derivations in the logic are universal in nature, where a derivation in the logic corresponds with denying all possible paths between source and destination.

In order to achieve this, the logic uses an inductive approach to deriving conclusions. We have a set of inference rules that resemble base cases, where one can infer a denied rewrite of a frame or packet between two variables based solely on the network's topology or logical organization. We also have a set of inference rules that resemble inductive steps. These inductive steps allow us to take applications of basic rules, traverse the network's topology, and climb the network stack to infer negative rewrites of application-layer messages. We also have the ability to make assumptions about negative rewrites between interface variables. These assumptions can be discharged in an inductive step by instrumenting access control at another place in the network where they would be subsumed.

This chapter provides a detailed description of the access control logic developed in this research. Section 4.1 describes the basic rules developed and therefore the instrumentation available using this approach. Section 4.2 describes the concept of assumptions as it pertains to the access control logic, and how it provides flexibility in instrumenting the network. Section 4.3 describes inductive rules based on the network's topology. Section 4.4 describes inductive rules based on the network stack. Finally, Section 4.6 introduces the concepts of soundness and completeness and how they apply to the access control logic. The complete listing of inference rules in the access control logic is listed in Appendix B.

4.1 Basic Rules

The access control logic provides some inference rules that allow us to derive conclusions about negative rewrites between variables using information contained in the network mapping. This information may be about the physical topology, like whether an interface is directly connected to another interface, or about logical organization, such as whether a filter contains a certain IP address. As with a derivation using the network semantics, a derivation using the access control logic produces a constraint set that must be satisfied and applied to the network to realize the desired conclusion. The constraints generated in such a derivation are specifically generated with these basic rules.

Rules in the access control logic resemble rules in the network semantics. Take the following rule, named (LINK-FILTER-DSTIP) as an example:

$$\frac{\begin{array}{l} switch \notin (Dn).type \\ host \notin (Dn).type \\ p \in P \\ u \in (Dm).ifaces \\ p.dstip \in (Dn).ifaces[v].ifilter[dstip] \\ m \neq n \end{array}}{D, A \vdash (p, u_e) \not\rightarrow (p, v_i)}$$

The rule's conclusion is stated below the horizontal line. All conclusions in the logic take this form, where D is the network, A is the assumption set (to be introduced in Section 4.2), p is the PDU value in question, u_e is the interface variable serving as the source of the rewrite, and v_i is the destination variable. Requirements are expressed in the antecedents of the rule above the horizontal line.

This section provides details about the basic rules developed in this work. We begin with filter rules, which govern placement of packet filters in the network. We then describe the (TERMINAL) rule, which provides some reasoning about terminal nodes. We then describe address mismatch rules, which provide reasoning about denying rewrites based on mismatches between a packet or frame destination address and an interface's assigned address. Finally, we describe a rule governing forwarding table entries.

4.1.1 Filter Rules

The logic contains several rules for denying rewrites between variables by installing ingress or egress filters. These rules comprise the bulk of the basic rules defined in the logic. Basically, the filter rules state that a rewrite can be denied if some part of the datagram, specifically some piece of header information, is a member of a filter set on either the source interface or the destination interface.

Filter rules are named intuitively so that a rule's name identifies which piece of header information is filtered. For example, the (LINK-FILTER-DSTIP) rule captures requirements for denying a translation between variables by installing an ingress filter on the destination interface that filters based on a packet's destination IP address. This rule is listed above.

Notice the antecedents of the (LINK-FILTER-DSTIP) rule. They ensure that the destination device is neither a switch nor a host. This is because a switch can only filter based on frame header information, based on the provided network semantics, and because we do not trust a host's configuration to enforce access control. The rule also ensures that the destination IP address of the packet p is contained in the destination interface's ingress filter set. Figure 4.1 graphically depicts this rule. An IP packet gets blocked upon ingress at v. Notice that there is no specific relationship between u and v; the rule applies even with multiple intermediate nodes between the interfaces.

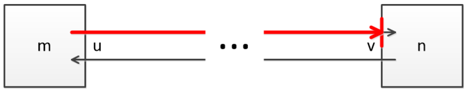

Figure 4.1: Depiction of (LINK-FILTER-DSTIP) rule application

4.1.2 (TERMINAL) Rule

The (TERMINAL) rule addresses the relationship between a terminal node and some other node in the network. It is the product of the following observation. The network semantics says that a terminal node – usually an end host – will not forward a PDU back through the interface through which it received the PDU. Therefore, if a device m directly connects via interface u to a terminal device n through interface v, any PDU sent from u to v will terminate at v. Therefore, the network's physical topology is enough to allow us to derive a conclusion about a negative rewrite from u to some other interface x. Since a PDU d from u always terminates at v, we know that d will never reach x from u. Formally:

$$\frac{\begin{array}{l}(D\,m).ifaces[u].direct = \{v\}\\ (D\,n).ifaces = \{v\}\\ v \neq x\end{array}}{D,A \vdash (d,u_e) \not\rightarrow (d,x_i)}$$

Figure 4.2 depicts the application of this rule. Notice that any traffic egressing from u terminates at v because there is no "U-turn" at v and because v is the only interface on n (as stated in the rule's second antecedent). Therefore, it is safe to say that traffic from u will never arrive at x, which is somewhere else in the network. Indeed, traffic from other interfaces on m may reach x, but that should be addressed elsewhere in the derivation. Our concern here is specifically traffic from u.

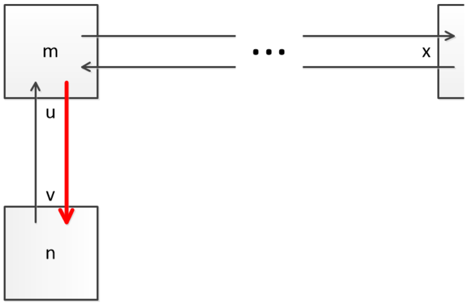

Figure 4.2: Depiction of (TERMINAL) rule application

4.1.3 Address Mismatch Rules

The logic includes several rules for denying rewrites based on address mismatches. These rules basically codify the fact that a network interface on a device other than a switch or router will not accept a PDU that is not addressed to that interface, which is expressed in the network semantics. We do not consider "promiscuous" interfaces, but the semantics could be extended with a promiscuous host interface, in which case deriving a negative outcome for such an interface could not rely on address mismatch. These rules basically state that we can deny rewriting a PDU to an interface's ingress variable if the destination address of the PDU does not match the interface's address. We include a rule named (IP-ADDRESS) for denying rewrites based on IP address mismatches and a rule named (HW-ADDRESS) for MAC address mismatches. The (IP-ADDRESS) rule is listed below:

$$\frac{\begin{array}{l} p \in P \\ u \in (Dm).ifaces \\ p.dstip \neq (Dn).ifaces[v].ipaddr \\ p.dstip \neq 255.255.255.255 \\ router \notin (Dn).type \\ switch \notin (Dn).type \end{array}}{D,A \vdash (p,u_e) \not\rightarrow (p,v_i)}$$

Several points should be explained here. First, we do not require $m \neq n$ or any other specific relationship between the devices or interfaces. As with the filter rules, this rule applies even across multiple intermediate nodes. Second, this rule is unsound if n is a router, hence the antecedent that n is not a router. Third, the rule is sound for limited broadcasts (255.255.255.255) only. A more refined rule would be also sound for directed broadcasts and would require checking the destination interface's subnet mask. Finally, soundness is mentioned here several times and is an important consideration throughout this system. We discuss this more in Section 4.6.

4.1.4 (SUBNET) Rule

The (SUBNET) rule allows denying rewrites between interfaces on the same device based on forwarding table entries. According to the network semantics, in order to route a packet

p from variable u_i to v_e on device n, $p.dstip$ must match $(Dn).ifaces[v].dest$. We leverage this requirement and say that we can deny rewriting p from u_i to v_e across m if $p.dstip$ does not match $(Dn).ifaces[v].dest$. This is formally captured below:

$$\frac{\begin{array}{l} u,v \in (Dn).ifaces \\ p \in P \\ (Dn).ifaces[v].netmask = mask \\ p.dstip\ \&\ mask \neq (Dn).ifaces[v].dest \\ router \in (Dn).type \end{array}}{D,A \vdash (p,u_i) \not\rightarrow (p,v_e)}$$

Figure 4.3 depicts this rule's application. Notice that p is not blocked as if we installed a packet filter on the egress interface. Instead, p is simply not routed from u to v because $p.dstip$ does not match the destination network of v.

4.2 Assumptions

The basic rules allow us to derive conclusions about negative rewrites based on information in the network mapping. For example, we can now deny a rewrite between two interface variables by placing a packet filter at either the source or destination. However, with these rules alone, we quickly see that our logic has a very limited ability to derive conclusions about negative rewrites across an entire network. We need mechanisms to allow us to reason about how applying these rules affects access control throughout the network. Furthermore, we need mechanisms that allow us to leverage topology in ensuring access control.

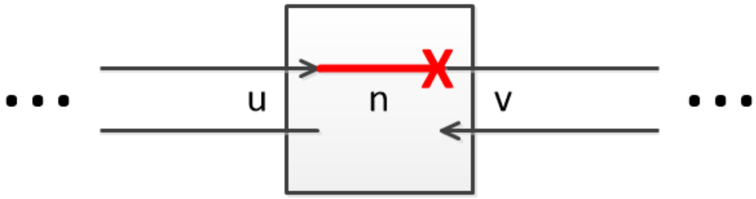

Figure 4.3: Depiction of (SUBNET) rule application

One of these mechanisms is the idea of assumptions about negative rewrites. In many cases, we find that we are obligated to prove a conclusion about a negative rewrite between two interface variables but that the basic rules alone do not provide any desirable options for doing so. In other words, we want to simply assume a negative rewrite so that we can continue the derivation in order to find another way to meet that obligation. Assumptions, when combined with the anti-forwarding rules that we introduce later, allow us the ability to reason about access control throughout the entire network.

Assumptions take the same form as negative rewrite operations and are held in a set that is carried through a derivation. For example, to state an assumption about a negative rewrite of packet p from interface u on device m to interface v on device n, we simply state that $(p, u_e) \not\rightarrow (p, v_i) \in A$.

To govern the use of assumptions in the logic, we provide two inference rules, (LINK-ASSUMPTION) and (DEVICE-ASSUMPTION). The (LINK-ASSUMPTION) rule simply states that we can derive a conclusion about a negative rewrite from an egress variable to an ingress variable if that negative rewrite is in the assumption set. Formally:

$$\frac{(d, u_e) \not\rightarrow (d, v_i) \in A}{D, A \vdash (d, u_e) \not\rightarrow (d, v_i)}$$

Notice that we do not restrict assumptions based on the format of the PDU or on the relationship between the two interface variables. This allows us freedom to make assumptions at any layer of the network stack and across any portion of the network. As we will see later, this freedom will translate into similar freedom in instrumenting the network.

The (DEVICE-ASSUMPTION) rule provides similar reasoning about negative rewrites from an ingress variable to an egress variable. This is used for assuming a negative rewrite within a device whereas the (LINK-ASSUMPTION) rule is used for assuming a negative rewrite across a link.

4.3 Anti-Forwarding Rules

With basic rules that govern how we actually instrument the network and with assumptions that allow us to carry proof obligations further along in the derivation, we have what we need to introduce the anti-forwarding rules. This set of rules allows us to reason inductively about negative rewrites across the network. With these rules, we also introduce the concept of discharging assumptions. By making an assumption, we create an obligation to reach that conclusion elsewhere in the derivation. By discharging an assumption, therefore, we instrument the network such that the obligation is rendered superfluous. Superfluous in this case means that the proof obligation is no longer necessary because another conclusion has been reached that subsumes that obligation.

The key to allowing us to reason about network-wide access control in this logic is the understanding that we are free to choose where in the network to discharge the assumptions that we make. Say we make assumption about a negative rewrite between a router and an end host. We could discharge that assumption immediately by instrumenting the router. We could also carry that assumption to an upstream router and instrument the network there in such a way as to render the assumption superfluous. In this way, we have the freedom we need to ensure access control throughout the network.

It is important to note that we must discharge all assumptions in a derivation before instrumenting the network. Remember that conclusions in the access control logic are based on both the network D and the assumption set A. If A is non-empty, there are proof obligations that have not been met. Consider a service specification in SOAC that includes the negative rewrite $(HTTP\_RQ, u_e) \not\rightharpoonup (HTTP\_RQ, v_i)$. Our goal, then, is to reach the conclusion $D, A \vdash (HTTP\_RQ, u_e) \not\rightharpoonup (HTTP\_RQ, v_i)$, for some assumption set A. We begin by making some assumptions about access control enforced at places in the network where needed to advance the derivation but may be undesirable in practice to implement there. These assumptions may be discharged upstream by virtue of network topology or by introducing access control at another place which subsumes them. In the latter case, they can be discharged, however, new assumptions arise due to introducing new access control elsewhere. In the end, A will contain assumptions that could not be discharged in the logic and therefore must be discharged outside the logic through instrumenting the network as they prescribe. This essentially produces a new network for which a derivation is now possible that ends with an empty A.

In this section, we introduce three inference rules that govern this inductive reasoning based on the network's physical topology. First, we describe the (FWD) rule, which allows us to carry assumptions across a router, a switch, or any type network bridge in order to discharge them further upstream. Second, we describe the (LINK-FWD) rule, which allows us to discharge certain assumptions at a network bridge by *instrumenting the network at the upstream link*, specifically by applying a basic rule to that link. Finally, we introduce the (DEVICE-FWD) rule, which governs discharging certain assumptions at a network bridge by *instrumenting the device itself.*

4.3.1 (FWD) Rule

The (FWD) rule allows us to carry assumptions across across a device without instrumenting that device. In doing so, we pass those assumptions further upstream so that we can meet those proof obligations elsewhere. Figure 4.4 shows a graphical depiction of this rule's application. Notice that we have an upstream device m with interface u that is directly connected to interface v of device n, which has three downstream interfaces w_1, w_2, and w_3. Suppose we want to prove that u_e cannot rewrite some variable, say x_i, which may be reachable through w_1, w_2, or w_3. For each downstream interface, we must first prove a negative rewrite between its egress variable and x_i (each depicted in the figure with a dark red "X"). If we can say that none of these egress variables is able reach x_1, even by assumption, we can then say that the u_e is not able to rewrite x_i (depicted in the figure with a bright red "X"). This rule is formally listed here:

$$\frac{\begin{array}{l} v \in (D\,n).ifaces \\ (D\,m).ifaces[u].direct = \{v\} \\ \forall w \in (D\,n).ifaces - \{v\}.\, D,A \vdash (d,w_e) \not\rightsquigarrow (d,x_i) \end{array}}{D,A \vdash (d,u_e) \not\rightsquigarrow (d,x_i)}$$

Notice that the assumption set A does not change as a result of this rule. This means that any assumption made in reaching the conclusions involving the downstream interfaces will be carried along and must be discharged elsewhere in the derivation.

4.3.2 (LINK-FWD) Rule

The (LINK-FWD) rule allows us to discharge assumptions at a device by instrumenting a the network at the upstream link. By instrumenting the upstream link, we can consider all proof obligations downstream to be superfluous. Figure 4.5 depicts this rule's application.

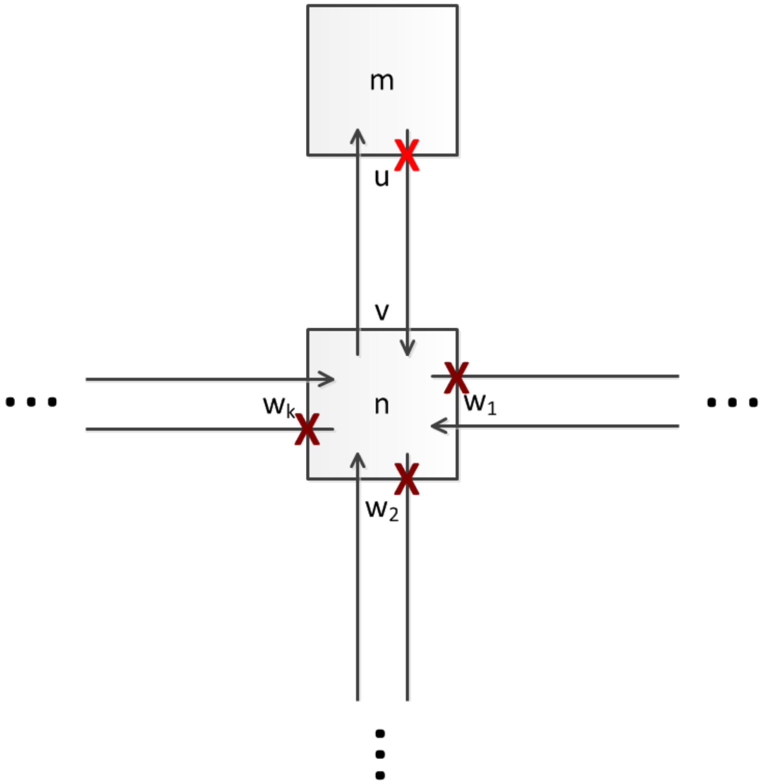

Figure 4.4: Depiction of (FWD) rule application

The required topology is the same as in the (FWD) rule. As with the (FWD) rule, we also reach the required conclusions about negative rewrites between the downstream interfaces and the destination interface. In accomplishing this, we build an assumption set $A \cup S$. Then, we instrument the upstream link and prove a corresponding negative rewrite. Any assumptions required to do so are contained in the set A. Having done these things, we can then discharge all assumptions in S. Formally:

$$\frac{\begin{array}{l} v \in (Dn).ifaces \\ (Dm).ifaces[u].direct = \{v\} \\ \forall w \in (Dn).ifaces - \{v\}.\, D, A \cup S \vdash (d, w_e) \not\rightarrow (d, x_i) \\ D, A \vdash (d, u_e) \not\rightarrow (d, v_i) \end{array}}{D, A \vdash (d, u_e) \not\rightarrow (d, x_i)}$$

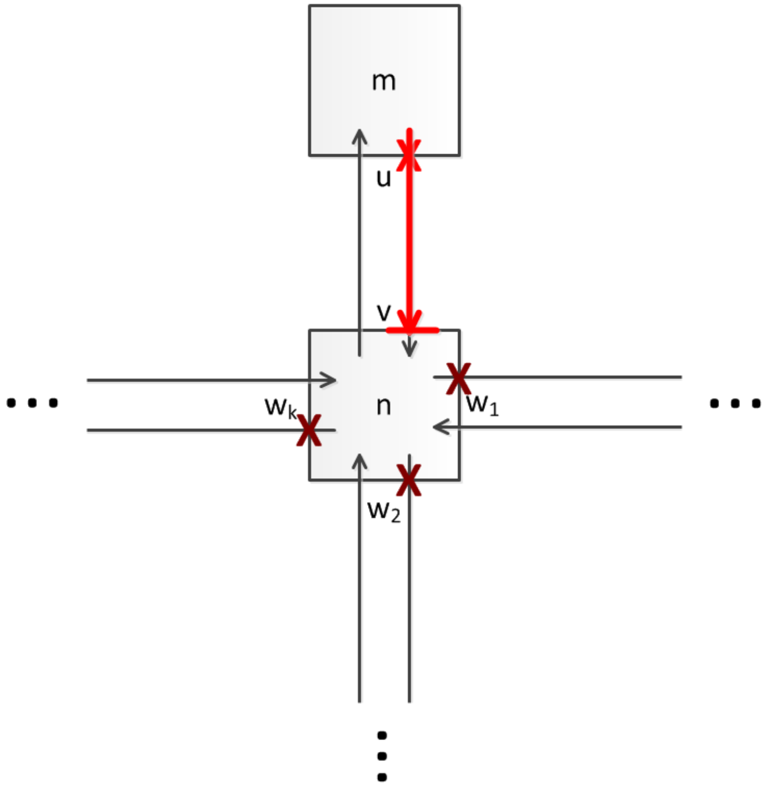

Figure 4.5: Depiction of (LINK-FWD) rule application

4.3.3 (DEVICE-FWD) Rule

With the (DEVICE-FWD) rule, we discharge assumptions not by instrumenting the upstream link but by instrumenting the device itself. We focus here on preventing rewrites across the device, usually by either egress filtering or constraints on the device's forwarding table. Again, the required topology is the same as the previous two rules. Also, the required assumptions involving downstream interfaces remain the same. Furthermore, we discharge in a similar fashion. Figure 4.6 depicts the application of this rule, and it is stated below:

$$\frac{\begin{array}{l} v \in (Dn).ifaces \\ (Dm).ifaces[u].direct = \{v\} \\ \forall w \in (Dn).ifaces - \{v\}.\, D,A \cup S \vdash (d,w_e) \not\rightsquigarrow (d,x_i) \\ \forall w \in (Dn).ifaces - \{v\}.\, D,A \vdash (d,v_i) \not\rightsquigarrow (d,w_e) \end{array}}{D,A \vdash (d,u_e) \not\rightsquigarrow (d,x_i)}$$

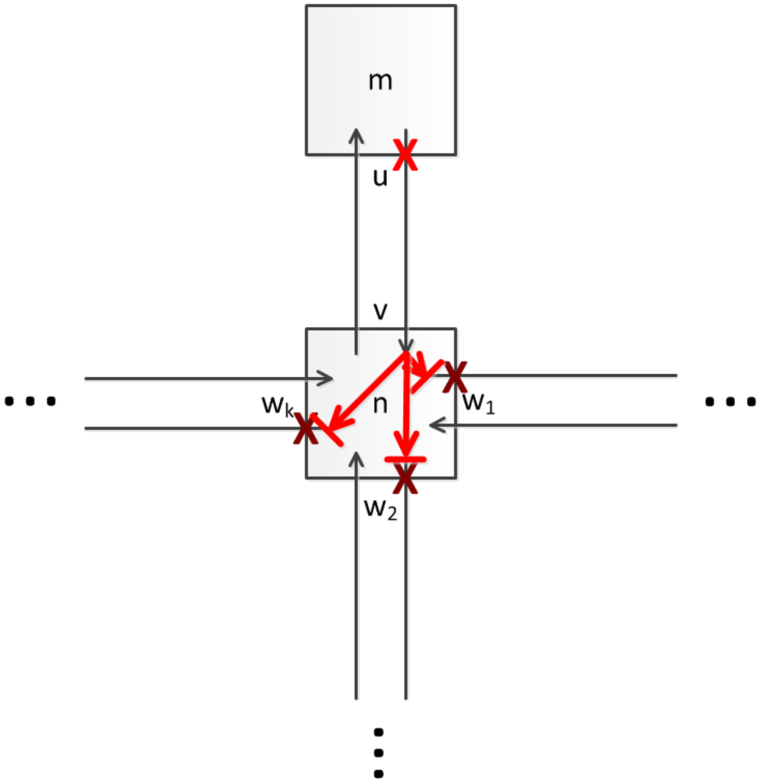

Figure 4.6: Depiction of (DEVICE-FWD) rule application

The difference between the (LINK-FWD) rule is in the fourth antecedent, which requires that we prove a negative rewrite from ingress variable v_i to every downstream egress variable on n.

4.4 Anti-Encapsulation Rules

With the combination of the basic rules, assumptions, and anti-forwarding rules, we have a system that allows us to derive conclusions about negative rewrites across the entire network. By using assumptions, we can choose where to instrument the network, and we can use the anti-forwarding rules to block all possible paths from source to destination.

However, the access control logic is still limited in that these rules only allow for reasoning about one particular layer of the network stack, usually a lower layer like the data-link or network layer. Remember that our goal is to develop a system that allows us to connect configuration at these lower levels to the uppermost layer where applications and services reside. Therefore, we need a system that gives us freedom to traverse not only the network topology but also the network stack.

To this end, the access control logic also provides inference rules that reason about negative rewrites at lower layers of the network stack and how those rewrites affect negative rewrites at higher layers. For example, the (HTTPS_RQ-ENCAP) rule governs the relationship between a negative rewrite of a HTTPS message and a negative rewrite of a TCP segment. The rule is listed below:

$$\frac{\begin{array}{l} s \in S \\ s.data = \alpha \\ \alpha = HTTPS\_RQ \\ D,A \vdash (s,u_e) \nrightarrow (s,v_i) \end{array}}{D,A \vdash (\alpha,u_e) \nrightarrow (\alpha,v_i)}$$

Notice that the rule establishes a relationship between the segment s and the HTTPS message α. This relationship, where $s.data = \alpha$, combined with the judgment that s at u cannot be rewritten as s at v, allows us to say that α at u also cannot be rewritten as α at v.

The access control logic contains anti-encapsulation rules to connect the data-link layer, network layer, transport layer, and application-layer protocols HTTPS and IRC. These application-layer protocols are not special; others could be introduced. We present these as representative protocols for the purpose of describing how applications are handled. Like the example rule, each of the anti-encapsulation inference rules provide a negative rewrite as a conclusion, a lower-layer negative rewrite as an antecedent, and some constraints that codify the relationship between the two negative rewrites.

4.5 A Decision Procedure for the Logic

A decision procedure for the logic has the property that if a derivation exists for a negative rewrite in the logic then the procedure will say “yes”, otherwise it will say “no”. To make such a procedure efficient, it is important to guide the construction of derivations in the logic as much as possible. The access control logic has been designed with this in mind. Specifically, it is “syntax-directed” [33], [34].

The anti-forwarding rules are designed such that a derivation for a negative rewrite begins at the destination and works its way back through the topology to the source. Consider the (FWD) rule as an example. Its third antecedent requires judgments of the form $D,A \vdash (d,w_e) \not\rightarrow (d,x_i)$ in order to reach the conclusion $D,A \vdash (d,u_e) \not\rightarrow (d,x_i)$, where w is on a device that is topologically closer to x than the device containing u. In other words, if we reach a conclusion of $D,A \vdash (d,w_e) \not\rightarrow (d,x_i)$, successive application of the (FWD) rule (or any anti-forwarding rule) yields $D,A \vdash (d,u_e) \not\rightarrow (d,x_i)$, where u is directly upstream of w. In this rule and throughout the logic, we see that successive conclusions are derived in terms of the same destination but in terms of incrementally more distant sources. This makes the access control logic effectively syntax-directed.

While syntax-directedness guides construction, there still remains much latitude for the decision procedure to choose from among many different derivations of the same negative-rewrite conclusion. While it is somewhat useful to know that if a negative rewrite deduction exists, a decision procedure can say so, we would like to know which instrumentations different derivations prescribe for the network. Ideally, a procedure would compute the "best" derivation according to some heuristics governing network traffic, performance, forwarding table size, etc. Such a procedure is beyond the scope of this thesis and is an area of future work.

4.6 Soundness and Completeness

There is a relationship between the access control logic and the semantics, and it is captured by the notions of soundness and completeness of the logic [34], [35]. In general, soundness and completeness describe how accurately a logical system represents reality. To say that a logical system is sound is to say that, if a statement can be shown in the system, it is indeed true in reality. Conversely, if any statement that holds in reality can be shown in the system, the system is said to be complete. For our purpose, reality is the network semantics, which codifies widespread, standards-based beliefs about how network devices behave.

We can express soundness of the access control logic in the following way. For any negative rewrite of the form $(d,u_e) \not\rightarrow (d,v_i)$ and for a given network D, if we derive $D,A \vdash (d,u_e) \not\rightarrow (d,v_i)$ in the logic, instrumenting D according to A (by apply basic rules like filtering and subnetting) yields a network D' such that the corresponding positive rewrite $(d,u_e) \rightarrow (d,v_i)$ cannot be derived in the semantics with respect to D'. Note that if A is empty, D needs no instrumentation, and $D' = D$. In other words, if we derive a conclusion about a negative rewrite in the logic, the resulting instrumentation will render it impossible to derive a conclusion about the corresponding positive rewrite in the semantics. Ensuring the soundness of the access control logic is critical to ensuring effective access control in the network.

Completeness says that, if there is a positive rewrite that is impossible to prove in the network semantics, then there exists a derivation in the logic to reach a conclusion about the corresponding negative rewrite. Completeness, for our purposes, is not as critical to the effectiveness of the logic as is soundness. If the logic is not complete, it is simply not leveraging all available capabilities to ensure access control. However, if the logic is not sound, it is not considering all possible paths, thereby falsely asserting access control.

Here, it is important to reiterate that the logic's soundness and completeness is measured against the network semantics, not against current networking capabilities. Consider, for example, an intrusion detection system (IDS) that operates in promiscuous mode, capturing all network traffic traversing a link. If this behavior were defined in the network semantics, then the address mismatch rules as written would be unsound because they falsely assert negative rewrites. However, we do not capture this behavior in the network semantics used in this thesis, so the behavior cannot be used to judge the logic as unsound.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Scenarios

USMC doctrine [36], [37] specifies that a USMC field artillery battalion typically consists of three firing batteries, each operating six 155-millimeter howitzers, and a headquarters battery that provides command, control, and support for the unit. To facilitate command and control, communications planners [38] typically establish a tactical data network throughout the battalion.

Figure 5.1 depicts this typical network as deployed in operational environments. In this typical network, each battery operates on a switched LAN that is connected to an IP router and then to a radio transceiver. The battalion headquarters also operates on a switched LAN that may contain some servers. When the battalion is operating as part of a larger unit, the battalion headquarters will connect to that unit, and the battalion will use those services. However, when the battalion is operating independently, the headquarters will provide the necessary services.

In some cases, an Ethernet mesh may replace radio systems as the network's backbone. A battalion may use this approach in conducting digital communications exercises, where the unit establishes this network test its digital command and control abilities. A battalion may also take this approach if its firing batteries are located in close proximity.

Figure 5.2 depicts the physical topology of a typical field artillery battalion's network with an Ethernet mesh backbone. This network lends itself well to the SOAC framework for two reasons. First, security requirements in this network are often fine-grained. By fine-grained, we mean that two hosts on a LAN may require access to different services; one host may be granted access to a server when another host on the same LAN may be denied access to the same server. Second, security requirements are often expressed in terms of services.

Take a battalion's mission planning process as an example. A battalion commander often expresses communications requirements in the following way. Some service will serve as the primary means of command and control and these units will use it, some other service

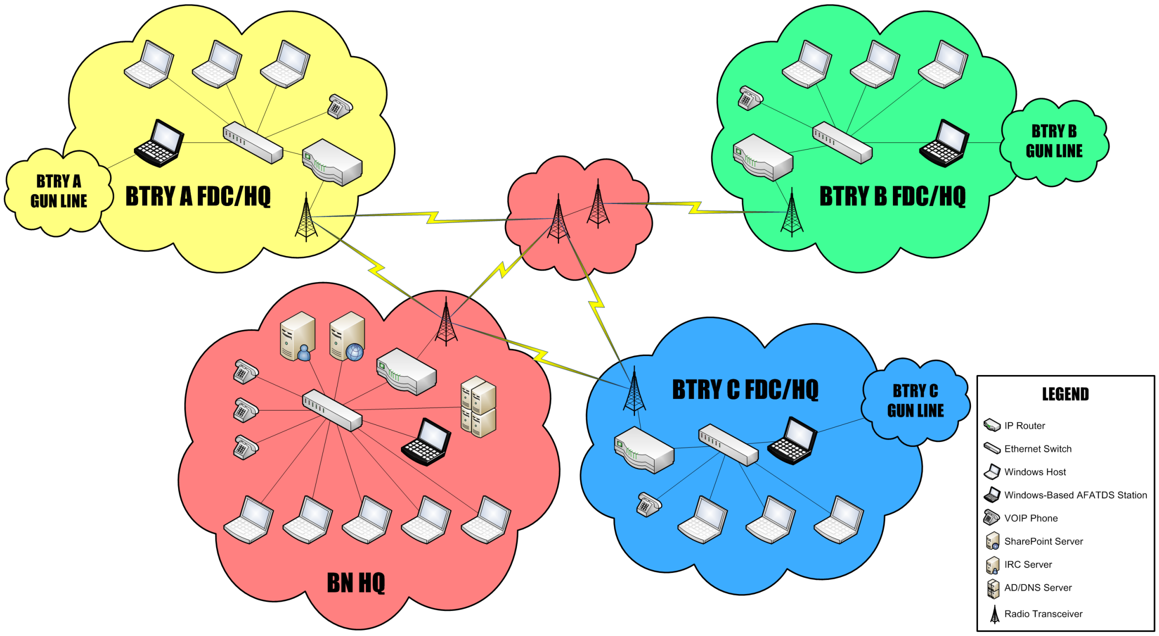

Figure 5.1: Common Marine Corps field artillery battalion data network

will serve as the secondary means and these units will use it, and so on. The battalion communications officer will then build a plan to support those requirements using the data network. With SOAC, those requirements can be used directly as input for developing the network configuration.

This chapter describes three scenarios where we use the SOAC framework to achieve an access control strategy by systematically generating constraints on the network. In the first scenario, we restrict access to the battalion's SharePoint server to only hosts in the battalion headquarters, showing how we take a simple service specification and apply our framework. In the second scenario, we maintain the SharePoint server access restriction while granting a host in Battery B access to the battalion's IRC server, introducing conflicting derivations that we resolve. Finally, we adjust the topology and introduce a service specification with an unresolvable conflict to show how our approach handles such a case.

5.1 First Scenario: Basic Use

Let D represent the network depicted in Figure 5.2. Consider a scenario where we must limit SharePoint access to hosts in the battalion headquarters. More specifically, we must

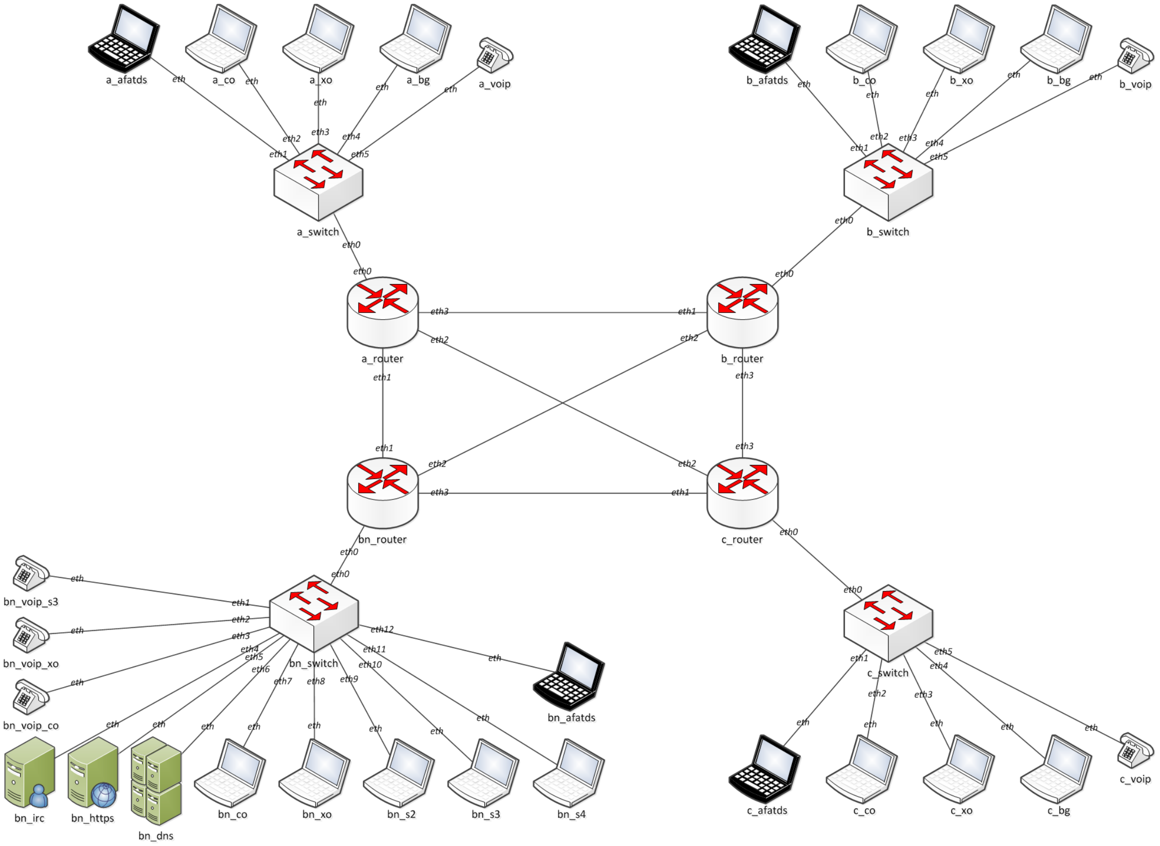

Figure 5.2: Physical network topology for common Marine Corps field artillery battalion data network

ensure that all hosts located in the battalion headquarters – namely, bn_co, bn_xo, bn_s2, bn_s3, and bn_s4 – can access the SharePoint server, and we must deny access to all other hosts in the network.

This requirement translates to several conclusions to be derived using either the network semantics or the access control logic. These conclusions are listed in Figure 5.3. We look at two of these desired outcomes in detail. First, we consider the desired conclusion $D \vdash (HTTPS\_RQ, bn\_co\_eth_e) \to (HTTPS\_RQ, bn\_https\_eth_i)$, which we use the network semantics to derive. Second, we consider $D, \emptyset \vdash (HTTPS\_RQ, a\_co\_eth_e) \not\to (HTTPS\_RQ, bn\_https\_eth_i)$, which we use the access control logic to derive.

$$
\begin{array}{l}
D \vdash (HTTPS\_RQ, bn\_co\_eth_e) \rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D \vdash (HTTPS\_RQ, bn\_xo\_eth_e) \rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D \vdash (HTTPS\_RQ, bn\_s2\_eth_e) \rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D \vdash (HTTPS\_RQ, bn\_s3\_eth_e) \rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D \vdash (HTTPS\_RQ, bn\_s4\_eth_e) \rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, a\_co\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, a\_xo\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, a\_bg\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, b\_co\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, b\_xo\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, b\_bg\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, c\_co\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, c\_xo\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
D, \emptyset \vdash (HTTPS\_RQ, c\_bg\_eth_e) \not\rightarrow (HTTPS\_RQ, bn\_https\_eth_i) \\
\ldots
\end{array}
$$

Figure 5.3: Service specification excerpt, first scenario

5.1.1 Positive Derivation

Consider the first desired conclusion, where an HTTPS request leaving from the battalion commander's host should translate to an HTTPS request arriving at the battalion's SharePoint server. The required derivation involves searching the network semantics for a path from the source incrementally through the network to the destination and then using the transitive property of the rewrite operation to reach the intended conclusion.

The derivation begins at the source with the (HTTPS_RQ-ENCAP) rule:

$$
\frac{\begin{array}{l}
s_1 = (\gamma_1, \gamma_2, \gamma_3, a) \\
a = HTTPS\_RQ \\
\gamma_1 = tcp \\
\gamma_2 \in (D\, bn\_co).portrange
\end{array}}{D \vdash (a, bn\_co\_eth_e) \rightarrow (s_1, bn\_co\_eth_e)}
$$

In applying this rule and reaching this conclusion, we generate some constraints on D that are necessary to allow the rewrite to occur. For example, $host \in (D\, bn\_co).type$ must be satisfied to reach this conclusion. As a result, fresh variables s_1, a, γ_1, γ_2, and γ_3 are generated. The derivation continues with the (PACKET-ENCAP) rule:

$$\frac{\begin{array}{l} p_1 = (\gamma_4, \gamma_5, s_1) \\ \gamma_6 = (D\, bn\_co).ifaces[bn\_co\_eth].netmask \\ \gamma_5 \ \&\ \gamma_6 = (D\, bn\_co).ifaces[bn\_co\_eth].dest \\ \gamma_4 = (D\, n).ifaces[w].ipaddr \\ \gamma_4 \notin (D\, bn\_co).ifaces[bn\_co\_eth].efilter[srcip] \\ \gamma_5 \notin (D\, bn\_co).ifaces[bn\_co\_eth].efilter[dstip] \end{array}}{D \vdash (s_1, bn\_co\_eth_e) \rightarrow (p_1, bn\_co\_eth_e)}$$

We generate more constraints on D as we reach this conclusion. We also generate more fresh variables (i.e., p_1, γ_4, γ_5, and γ_6). At this point, we also see constraints on the logical organization of D. For example, we constrain $(D\, bn\_co).ifaces[bn\_co\_eth].netmask$, which corresponds with the subnet mask of the interface bn_co_eth.

The derivation continues in this way, where we search for a path from the source incrementally through the network to the destination. For the sake of brevity, several steps are omitted here. As the derivation progresses, we generate more constraints. At the destination, the (HTTPS_RQ-DECAP) rule applies:

$$\frac{\gamma_2 = 443}{D \vdash (s_1, bn\_https\_eth_i) \rightarrow (a, bn\_https\_eth_i)}$$

Upon reaching the destination, we see the transitive property of the rewrite operation help reach the final conclusion. The (TRANS) is used for this purpose:

$$\frac{\begin{array}{c} D \vdash (p_1, bn\_https\_eth_i) \rightarrow (s_1, bn\_https\_eth_i) \\ D \vdash (s_1, bn\_https\_eth_i) \rightarrow (a, bn\_https\_eth_i) \end{array}}{D \vdash (p_1, bn\_https\_eth_i) \rightarrow (a, bn\_https\_eth_i)}$$

The (TRANS) rule essentially connects the individual conclusions together. Both premises used here are conclusions reached earlier. Also, notice that no new constraints are generated after applying the (TRANS) rule. This is expected since we are simply using the transitive property of the rewrite operation to complete the derivation. We continue with successive applications of the (TRANS) rule, working backward from the destination to the source, until we are able to derive our desired conclusion:

$$\frac{\begin{array}{c} D \vdash (a, bn\_co\_eth_e) \rightarrow (s_1, bn\_co\_eth_e) \\ D \vdash (s_1, bn\_co\_eth_e) \rightarrow (a, bn\_https\_eth_i) \end{array}}{D \vdash (a, bn\_co\_eth_e) \rightarrow (a, bn\_https\_eth_i)}$$

With this derivation, we effectively prove, based on the network semantics, that the battalion commander's host can send an HTTPS request to the SharePoint server. Several constraints on *D* are generated as a result. The complete set of constraints from the derivation is listed in Figure 5.4. By satisfying these constraints, we can produce a constrained *D* that corresponds with a logical organization that can be applied to the network's devices. This logical organization can realize our service-level goals.

5.1.2 Negative Derivation

Now consider the second desired conclusion, where an HTTPS request leaving from the Battery A commander's host must not translate to an HTTPS request at the battalion's SharePoint server. Note that *D* has already been constrained as a result of the previous derivation.

We begin at the destination and work our way back to the source. We also choose to focus on the *bn_router* device for any instrumentation. Therefore, we use the (ASSUMPTION) rule to eliminate any instrumentation needs downstream of *bn_router*:

$$\frac{\begin{array}{l} p_2 = (\gamma_9, \gamma_{10}, s_2) \\ A_1 = \{(p_2, bn\_router\_eth0_e) \nrightarrow (p_2, bn\_https\_eth_i)\} \end{array}}{D, A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \vdash (p_2, bn\_router\_eth0_e) \nrightarrow (p_2, bn\_https\_eth_i)}$$

$$\frac{A_2 = \{(p_2, bn\_router\_eth2_e) \nrightarrow (p_2, bn\_https\_eth_i)\}}{D, A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \vdash (p_2, bn\_router\_eth2_e) \nrightarrow (p_2, bn\_https\_eth_i)}$$

$$\frac{A_3 = \{(p_2, bn\_router\_eth3_e) \nrightarrow (p_2, bn\_https\_eth_i)\}}{D, A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \vdash (p_2, bn\_router\_eth3_e) \nrightarrow (p_2, bn\_https\_eth_i)}$$

$$
\begin{array}{l}
a = \mathit{HTTPS\_RQ} \\
s_1 = (\gamma_1, \gamma_2, \gamma_3, a) \\
\gamma_1 = tcp \\
\gamma_2 \in (D\,bn\_co).portrange \\
p_1 = (\gamma_4, \gamma_5, s_1) \\
\gamma_6 = (D\,bn\_co).ifaces[bn\_co\_eth].netmask \\
\gamma_5 \;\&\; \gamma_6 = (D\,bn\_co).ifaces[bn\_co\_eth].dest \\
\gamma_4 = (D\,bn\_co).ifaces[bn\_co\_eth].ipaddr \\
\gamma_4 \notin (D\,bn\_co).ifaces[bn\_co\_eth].efilter[srcip] \\
\gamma_5 \notin (D\,bn\_co).ifaces[bn\_co\_eth].efilter[dstip] \\
f_1 = (\gamma_7, \gamma_8, p_1) \\
\gamma_7 = (D\,bn\_co).ifaces[bn\_co\_eth].hwaddr \\
\gamma_7 \notin (D\,bn\_co).ifaces[bn\_co\_eth].efilter[srchw] \\
\gamma_8 \notin (D\,bn\_co).ifaces[bn\_co\_eth].efilter[dsthw] \\
\gamma_7 \notin (D\,bn\_switch).ifaces[bn\_switch\_eth7].ifilter[srchw] \\
\gamma_8 \notin (D\,bn\_switch).ifaces[bn\_switch\_eth7].ifilter[dsthw] \\
\gamma_7 \notin (D\,bn\_switch).ifaces[bn\_switch\_eth5].efilter[srchw] \\
\gamma_8 \notin (D\,bn\_switch).ifaces[bn\_switch\_eth5].efilter[dsthw] \\
\gamma_8 = (D\,bn\_https).ifaces[bn\_https\_eth].hwaddr \\
\gamma_7 \notin (D\,bn\_https).ifaces[bn\_https\_eth].ifilter[srchw] \\
\gamma_8 \notin (D\,bn\_https).ifaces[bn\_https\_eth].ifilter[dsthw] \\
\gamma_4 \notin (D\,bn\_https).ifaces[bn\_https\_eth].ifilter[srcip] \\
\gamma_5 \notin (D\,bn\_https).ifaces[bn\_https\_eth].ifilter[dstip] \\
\gamma_4 = (D\,bn\_https).ifaces[bn\_https\_eth].ipaddr \\
(\gamma_1, \gamma_2) \notin (D\,bn\_https).ifaces[bn\_https\_eth].ifilter[srcport] \\
(\gamma_1, \gamma_3) \notin (D\,bn\_https).ifaces[bn\_https\_eth].ifilter[dstport] \\
\gamma_2 = 443
\end{array}
$$

Figure 5.4: Result of positive derivation, first scenario

Notice that we assume negative conclusions about negative rewrites from every interface on bn_router to bn_https_eth except from bn_router_eth1, which is directly connected to the upstream a_router_eth1 interface. This prepares us to apply the inductive (DEVICE-FWD) rule to bn_router. The (DEVICE-FWD) rule requires us to reach conclusions about negative rewrites from every egress variable on a device, except the variable associated with the upstream interface. It then requires us to reach conclusions about negative rewrites between the interfaces on the bn_router device. We use rules like (DEVICE-FILTER-DSTIP), which introduces a packet filter matching a packet's destination IP address, and (SUBNET), which constrains the device's forwarding table. Once we reach a conclusion about a negative rewrite between two of these interfaces, we then apply the rule to discharge the appropriate assumptions as superfluous. This part of the derivation begins with application of the (SUBNET) rule in order to discharge A_2:

$$\frac{\begin{array}{l}\gamma_{11} = (D\, bn\_router).ifaces[bn\_router\_eth2].netmask \\ \gamma_{10}\ \&\ \gamma_{11} \neq (D\, bn\_router).ifaces[bn\_router\_eth2].dest\end{array}}{D, A_4 \cup A_5 \vdash (p_2, bn\_router\_eth1_i) \not\rightarrow (p_2, bn\_router\_eth2_e)}$$

Notice that constraints on the device's forwarding table appear here. These constraints, when satisfied, will be used to instrument D. We use the (SUBNET) rule again to help discharge A_3. This leaves A_1, which is treated with the (DEVICE-FILTER-DSTIP) rule:

$$\frac{\gamma_{10} \in (D\, bn\_router).ifaces[bn\_router\_eth0].efilter[dstip]}{D, A_4 \cup A_5 \vdash (p_2, bn\_router\_eth1_i) \not\rightarrow (p_2, bn\_router\_eth0_e)}$$

We have now reached all conclusions required by the (DEVICE-FWD) rule. This rule is applied to discharge assumptions A_1, A_2, and A_3, allowing us to continue the derivation further upstream. We apply the rule here:

$$\frac{\begin{array}{l}D, A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \vdash (p_2, bn\_router\_eth0_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D, A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \vdash (p_2, bn\_router\_eth2_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D, A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \vdash (p_2, bn\_router\_eth3_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D, A_4 \cup A_5 \vdash (p_2, bn\_router\_eth1_i) \not\rightarrow (p_2, bn\_router\_eth0_e) \\ D, A_4 \cup A_5 \vdash (p_2, bn\_router\_eth1_i) \not\rightarrow (p_2, bn\_router\_eth2_e) \\ D, A_4 \cup A_5 \vdash (p_2, bn\_router\_eth1_i) \not\rightarrow (p_2, bn\_router\_eth3_e)\end{array}}{D, A_4 \cup A_5 \vdash (p_2, a\_router\_eth1_e) \not\rightarrow (p_2, bn\_https\_eth_i)}$$

At this point, we find that the packet p_2 cannot be rewritten from interface *a_router_eth*1 to *bn_https_eth*. In effect, we block every path through the network for this packet to travel from *a_router_eth*1 to *bn_https_eth*.

In this scenario, however, there are other paths from the source to the destination. These paths must be blocked as well, so we continue the derivation. We treat *a_router* with a similar strategy as before. First, we make some assumptions as we did previously. Second, we apply rules that introduce an instrumentation on the target device. Third, we apply an anti-forwarding rule to discharge the appropriate assumptions. This strategy allows us to reach the conclusion $D,\emptyset \vdash (p_2, a\_switch\_eth0_e) \not\rightarrow (p_2, bn\_https\_eth_i)$.

At this point, the (TERMINAL) rule becomes useful. We use the rule to reach negative conclusions about devices on the same LAN as the source by showing that each port on *a_switch* terminates with a device that is not the destination. We apply the rule to all interfaces on the switch besides the interface connected to *a_co* to prepare for using the (FWD) rule. Recall that we must reach conclusions about negative rewrites from all egress variables on the target device to the destination. Having accomplished this with the (TERMINAL) rule, the derivation continues by applying the (FWD) rule:

$$\frac{\begin{array}{l} D,\emptyset \vdash (p_2, a\_switch\_eth0_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D,\emptyset \vdash (p_2, a\_switch\_eth1_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D,\emptyset \vdash (p_2, a\_switch\_eth3_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D,\emptyset \vdash (p_2, a\_switch\_eth4_e) \not\rightarrow (p_2, bn\_https\_eth_i) \\ D,\emptyset \vdash (p_2, a\_switch\_eth5_e) \not\rightarrow (p_2, bn\_https\_eth_i) \end{array}}{D,\emptyset \vdash (p_2, a\_co\_eth_e) \not\rightarrow (p_2, bn\_https\_eth_i)}$$

Therefore, we can successfully derive the conclusion that, given only the mapping D, the packet p_2 at *a_co_eth* cannot be rewritten as p_2 at *bn_https_eth*. However, we need to reach a conclusion in terms of the application-layer *HTTPS_RQ*. Therefore, we continue the derivation using the (SEG-ENCAP) and (HTTPS_RQ-ENCAP) rules to arrive at the final conclusion of $D,\emptyset \vdash (a, a\_co\_eth_e) \not\rightarrow (a, bn\_https\_eth_i)$. Now, similar to the positive case, we show that, given only the mapping D, the message *HTTP_RQ* at *a_co_eth* cannot be rewritten as *HTTP_RQ* at *bn_https_eth*. In plain English, we show that the battery commander's host cannot send an HTTPS request to the SharePoint server. Several constraints are generated as a result and are listed in Figure 5.5. By combining this set with the one generated in the positive derivation, we will have a constraint set whose satisfying substitution will allow us to configure the network to realize both the positive and negative service-level goals.

$$
\begin{array}{l}
p_2 = (\gamma_9, \gamma_{10}, s_2) \\
A_1 = \{(p_2, \mathit{bn\_router\_eth0}_e) \not\rightarrow (p_2, \mathit{bn\_https\_eth}_i)\} \\
A_2 = \{(p_2, \mathit{bn\_router\_eth2}_e) \not\rightarrow (p_2, \mathit{bn\_https\_eth}_i)\} \\
A_3 = \{(p_2, \mathit{bn\_router\_eth3}_e) \not\rightarrow (p_2, \mathit{bn\_https\_eth}_i)\} \\
\gamma_{11} = (D\,\mathit{bn\_router}).\mathit{ifaces}[\mathit{bn\_router\_eth2}].\mathit{netmask} \\
\gamma_{10} \,\&\, \gamma_{11} \neq (D\,\mathit{bn\_router}).\mathit{ifaces}[\mathit{bn\_router\_eth2}].\mathit{dest} \\
\gamma_{12} = (D\,\mathit{bn\_router}).\mathit{ifaces}[\mathit{bn\_router\_eth3}].\mathit{netmask} \\
\gamma_{10} \,\&\, \gamma_{12} \neq (D\,\mathit{bn\_router}).\mathit{ifaces}[\mathit{bn\_router\_eth3}].\mathit{dest} \\
\gamma_{10} \in (D\,\mathit{bn\_router}).\mathit{ifaces}[\mathit{bn\_router\_eth0}].\mathit{efilter}[\mathit{dstip}] \\
A_4 = \{(p_2, \mathit{a\_router\_eth2}_e) \not\rightarrow (p_2, \mathit{bn\_https\_eth}_i)\} \\
A_5 = \{(p_2, \mathit{a\_router\_eth3}_e) \not\rightarrow (p_2, \mathit{bn\_https\_eth}_i)\} \\
\gamma_{13} = (D\,\mathit{a\_router}).\mathit{ifaces}[\mathit{a\_router\_eth2}].\mathit{netmask} \\
\gamma_{10} \,\&\, \gamma_{13} \neq (D\,\mathit{a\_router}).\mathit{ifaces}[\mathit{a\_router\_eth2}].\mathit{dest} \\
\gamma_{14} = (D\,\mathit{a\_router}).\mathit{ifaces}[\mathit{a\_router\_eth2}].\mathit{netmask} \\
\gamma_{10} \,\&\, \gamma_{14} \neq (D\,\mathit{a\_router}).\mathit{ifaces}[\mathit{a\_router\_eth3}].\mathit{dest} \\
\gamma_{10} = (D\,\mathit{bn\_https}).\mathit{ifaces}[\mathit{bn\_https\_eth}].\mathit{ipaddr} \\
s_2 = (\gamma_{15}, \gamma_{16}, \gamma_{17}, a) \\
\gamma_{15} = \mathit{tcp} \\
\gamma_{17} = 443 \\
a = \mathit{HTTPS\_RQ}
\end{array}
$$

Figure 5.5: Result of negative derivation, first scenario

5.2 Second Scenario: Resolvable Conflicts

Consider another scenario where, though firing battery commanders are denied access to the battalion's SharePoint server, they are allowed access to the IRC server for tactical chat and messaging. Though we must derive several conclusions to meet this requirement, we look at only two in order to illustrate the approach. First, we consider the desired positive conclusion $D \vdash (\mathit{IRC\_MSG}, \mathit{b\_co\_eth}_e) \rightarrow (\mathit{IRC\_PRIVMSG}, \mathit{bn\_irc\_eth}_i)$, which we derived with the network semantics. Second, we consider $D, \emptyset \vdash (\mathit{HTTPS\_RQ}, \mathit{b\_co\_eth}_e) \not\rightarrow (\mathit{HTTPS\_RQ}, \mathit{bn\_https\_eth}_i)$, which derive with the access control logic.

5.2.1 Positive Derivation

Consider the first desired conclusion; we seek to show that an IRC private message at the Battery B commander's host would translate to an IRC private message at the battalion's chat server. Our approach here is similar to our approach in the previous scenario. We begin at the source and work our way to the destination.

The derivation specifically begins with the (IRC_PRIVMSG-ENCAP) rule:

$$\frac{\begin{array}{l} a_1 = \mathit{IRC\_PRIVMSG} \\ s_1 = (\gamma_1, \gamma_2, \gamma_3, a_1) \\ \gamma_1 = tcp \\ \gamma_2 \in (D\,b\_co).portrange \end{array}}{D \vdash (a_1, b\_co\_eth_e) \to (s_1, b\_co\_eth_e)}$$

The derivation continues here similar to the positive derivation in the previous example. In this scenario, however, our path takes us to the device *b_router*. To show the routing of the IP packet across this router, we apply the (ROUTER) rule:

$$\frac{\begin{array}{l} \gamma_9 = (D\,b\_router).ifaces[b\_router\_eth2].netmask \\ \gamma_5 \neq 255.255.255.255 \\ \gamma_5 \neq \neg\gamma_9 \mid (D\,b\_router).ifaces[b\_router\_eth2].dest \\ \gamma_5 \ \&\ \gamma_9 = (D\,b\_router).ifaces[b\_router\_eth2].dest \\ \gamma_4 \notin (D\,b\_router).ifaces[b\_router\_eth2].efilter[srcip] \\ \gamma_5 \notin (D\,b\_router).ifaces[b\_router\_eth2].efilter[dstip] \end{array}}{D \vdash (p_1, b\_router\_eth0_i) \to (p_1, b\_router\_eth2_e)}$$

Notice here that several constraints are placed on subnet mask, destination IP address, and forwarding table. These constraints, when satisfied, ensure that the router forwards the packet based on matching network prefix to the appropriate interface.

With respect to the network topology, we find ourselves now inside the network's Ethernet backbone. Our derivation, however, continues in the familiar way, with encapsulations followed by transmissions followed by decapsulations. We apply the (ROUTER) rule again once we reach the battalion's router, and then we continue through the battalion's LAN.

Having reached our destination, the device *bn_irc*, we begin to apply rules to decapsulate the frame. We begin with the (FRAME-DECAP) rule and conclude by applying the

(IRC_PRIVMSG-DECAP) rule:

$$\frac{\gamma_3 = 6667}{D \vdash (s_1, bn\_irc\_eth_i) \rightarrow (a_1, bn\_irc\_eth_i)}$$

Similar to the previous scenario, the derivation continues backward toward the source with the (TRANS) rule successively applied until we reach the ultimate conclusion of $D \vdash (a_1, b\_co\_eth_i) \rightarrow (a_1, bn\_irc\_eth_i)$. Again, we use this derivation to prove that an IRC private message can be sent from the Battery B commander's host to the battalion's chat server.

In doing so, we generate more constraints to apply to the mapping D. Our constraint set here is much larger than the set generated in the previous scenario, due to the increased length of the path through the network. Figure 5.6 shows a portion of the constraint set. A satisfying substitution can then be applied to the network's devices to realize our service-level goals here.

5.2.2 Negative Derivation

Now consider the second desired conclusion, where the battery commander should not have access to the battalion's SharePoint server. We take an approach similar to the approach in the previous scenario. We begin at the destination and work our way back to the source.

In this scenario, though, we decide to focus on the device *b_router* for instrumenting the network. We begin by making assumptions about connectivity between *bn_https_eth* and the interfaces of *b_router*. We use the (ASSUMPTION) rule to declare three assumptions like the following:

$$\frac{\begin{array}{l} p_2 = (\gamma_{15}, \gamma_{16}, s_2) \\ A_1 = (p_2, b\_router\_eth2_e) \not\rightarrow (p_2, bn\_https\_eth_i) \end{array}}{D, A_1 \cup A_2 \cup A_3 \vdash (p_2, b\_router\_eth2_e) \not\rightarrow (p_2, bn\_https\_eth_i)}$$

We decide to leverage the (LINK-FWD) rule to discharge all three assumptions at once. In order to do that, we introduce constraints on the link directly upstream from the

$$
\begin{aligned}
&a_1 = \mathit{IRC\_PRIVMSG}\\
&s_1 = (\gamma_1, \gamma_2, \gamma_3, a_1)\\
&\gamma_1 = tcp\\
&\ldots\\
&\gamma_8 = (D\,b\_router).ifaces[b\_router\_eth0].hwaddr\\
&\gamma_7 \notin (D\,b\_router).ifaces[b\_router\_eth0].ifilter[srchw]\\
&\gamma_8 \notin (D\,b\_router).ifaces[b\_router\_eth0].ifilter[dsthw]\\
&\ldots\\
&(\gamma_1, \gamma_2) \notin (D\,bn\_irc).ifaces[bn\_irc\_eth].ifilter[srcport]\\
&(\gamma_1, \gamma_2) \notin (D\,bn\_irc).ifaces[bn\_irc\_eth].ifilter[dstport]\\
&\gamma_3 = 6667
\end{aligned}
$$

Figure 5.6: Excerpt of fresh variables generated during positive derivation, second scenario

router. Specifically, we decide to introduce MAC address filtering on the router's interface b_router_eth0, which is connected to the Battery B LAN. We therefore apply the (LINK-FILTER-DSTHW) rule:

$$
\frac{\begin{array}{l} f_4 = (\gamma_{17}, \gamma_{18}, p_2) \\ \gamma_{18} \in (D\,b\_router).ifaces[b\_router\_eth0].ifilter[dsthw] \end{array}}{D, \emptyset \vdash (f_4, b\_switch\_eth0_e) \not\rightsquigarrow (f_4, b\_router\_eth0_i)}
$$

We then apply the (PKT-ENCAP) rule, which is what we need in order to apply the (LINK-FWD) rule to discharge the assumptions:

$$
\frac{\begin{array}{l} D, A_1 \cup A_2 \cup A_3 \vdash (p_2, b\_router\_eth1_e) \not\rightsquigarrow (p_2, bn\_https\_eth_i) \\ D, A_1 \cup A_2 \cup A_3 \vdash (p_2, b\_router\_eth2_e) \not\rightsquigarrow (p_2, bn\_https\_eth_i) \\ D, A_1 \cup A_2 \cup A_3 \vdash (p_2, b\_router\_eth3_e) \not\rightsquigarrow (p_2, bn\_https\_eth_i) \\ D, \emptyset \vdash (p_2, b\_switch\_eth0_e) \not\rightsquigarrow (p_2, b\_router\_eth0_i) \end{array}}{D, \emptyset \vdash (p_2, b\_switch\_eth0_e) \not\rightsquigarrow (p_2, bn\_https\_eth_i)}
$$

Inside the Battery B LAN, we use the same strategy as the one in the previous scenario. The (TERMINAL) rule is used to show negative conclusions about the other devices on the LAN. Doing this allows us to continue the derivation and reach a conclusion about a negative rewrite between the source and destination. By using the (FWD) rule, we reach

$$
\begin{aligned}
&p_2 = (\gamma_{15}, \gamma_{16}, s_2)\\
&A_1 = (p_2, b\_router\_eth2_e) \not\rightarrow (p_2, bn\_https\_eth_i)\\
&A_2 = (p_2, b\_router\_eth1_e) \not\rightarrow (p_2, bn\_https\_eth_i)\\
&A_3 = (p_2, b\_router\_eth3_e) \not\rightarrow (p_2, bn\_https\_eth_i)\\
&f_4 = (\gamma_{17}, \gamma_{18}, p_2)\\
&\gamma_{18} \in (D\,b\_router).ifaces[b\_router\_eth0].ifilter[dsthw]\\
&\gamma_{18} = (D\,b\_router).ifaces[b\_router\_eth0].hwaddr\\
&\gamma_{16} = (D\,bn\_https).ifaces[bn\_https\_eth].ipaddr\\
&s_2 = (\gamma_{19}, \gamma_{20}, \gamma_{21}, a_2)\\
&\gamma_{19} = tcp\\
&\gamma_{21} = 443\\
&a_2 = HTTPS\_RQ
\end{aligned}
$$

Figure 5.7: First attempt, negative derivation, second scenario

the conclusion $D, \emptyset \vdash (p_2, b\_co\_eth_e) \not\rightarrow (p_2, bn\_https\_eth_i)$. We continue by using the (SEG-ENCAP) and (HTTPS_RQ-ENCAP) rules to arrive at the final conclusion of $D, \emptyset \vdash (a_2, b\_co\_eth_e) \not\rightarrow (a_2, bn\_https\_eth_i)$.

We find that this derivation generates constraints that contradict those generated in the positive derivation. By using unification, we find that the negative derivation yields the following constraint:

$$
\begin{aligned}
&(D\,b\_router).ifaces[b\_router\_eth0].hwaddr\\
&\qquad \in (D\,b\_router).ifaces[b\_router\_eth0].ifilter[dsthw]
\end{aligned}
$$

This contradicts a constraint generated in the positive derivation:

$$
\begin{aligned}
&(D\,b\_router).ifaces[b\_router\_eth0].hwaddr\\
&\qquad \notin (D\,b\_router).ifaces[b\_router\_eth0].ifilter[dsthw]
\end{aligned}
$$

It is clear, therefore, that the union of these two sets is unsatisfiable.

We know by intuition that this conflict can be resolved by trying a different derivation to reach the second conclusion. We can focus our efforts on a different device besides *b_router* by adjusting our assumptions, or we can simply adjust our instrumentation. We choose to adjust the instrumentation by filtering based on IP address rather than MAC address.

We restart our derivation, following the same strategy as before. We use the (LINK-FILTER-DSTIP) on *b_router* before applying (LINK-FWD):

$$\frac{\gamma_{16} \in (D\, b\_router).ifaces[b\_router\_eth0].ifilter[dstip]}{D, \emptyset \vdash (p_2, b\_switch\_eth0_e) \not\rightarrow (p_2, b\_router\_eth0_i)}$$

The derivation continues as before, eventually reaching the final conclusion. The result is listed in Figure 5.8. By inspection, we find that our conflict is resolved. The union of these two sets will result in a satisfiable constraint set, one whose satisfying substitution will allow us to configure the network to realize both the positive and negative service-level goals.

5.3 Third Scenario: Unresolvable Conflicts

In the previous scenarios, we successfully applied the SOAC approach to generate constraints on a network configuration for realizing service-level goals. In the first scenario, we did this without any conflict between our positive derivation and our negative derivation. In the second scenario, we found that, after completing the positive derivation, our first attempt at the negative derivation created a conflict and an unsatisfiable constraint set. We addressed this conflict in our second attempt at the negative derivation. We adjusted our instrumentation of the network, and we resolved the conflict.

We now introduce a scenario that presents an unresolvable conflict between positive and negative derivations. We adjust the network topology to correspond with the network in Figure 5.9. One physical device, *bn_server*, is providing both the SharePoint and IRC services for the network, which consists of a single router and two hosts, *c_co* and *c_xo*, connected via a switch. In order to access each service, hosts will send messages to the same interface – and therefore the same IP address – but to different ports. In this scenario, we must grant SharePoint access to the Battery C commander's host and IRC access to the Battery C executive officer's host. We must also deny SharePoint access to the executive officer's host and deny IRC access to the commander's host.

$$
\begin{aligned}
&A_1 = (p_2, b\_router\_eth2_e) \not\to (p_2, bn\_https\_eth_i)\\
&A_2 = (p_2, b\_router\_eth1_e) \not\to (p_2, bn\_https\_eth_i)\\
&A_3 = (p_2, b\_router\_eth3_e) \not\to (p_2, bn\_https\_eth_i)\\
&p_2 = (\gamma_{15}, \gamma_{16}, s_2)\\
&\gamma_{16} \in (D\, b\_router).ifaces[b\_router\_eth0].ifilter[dstip]\\
&\gamma_{16} = (D\, bn\_https).ifaces[bn\_https\_eth].ipaddr\\
&s_2 = (\gamma_{17}, \gamma_{18}, \gamma_{19}, a_2)\\
&\gamma_{17} = tcp\\
&\gamma_{19} = 443\\
&a_2 = HTTPS\_RQ
\end{aligned}
$$

Figure 5.8: Second attempt, negative derivation, second scenario

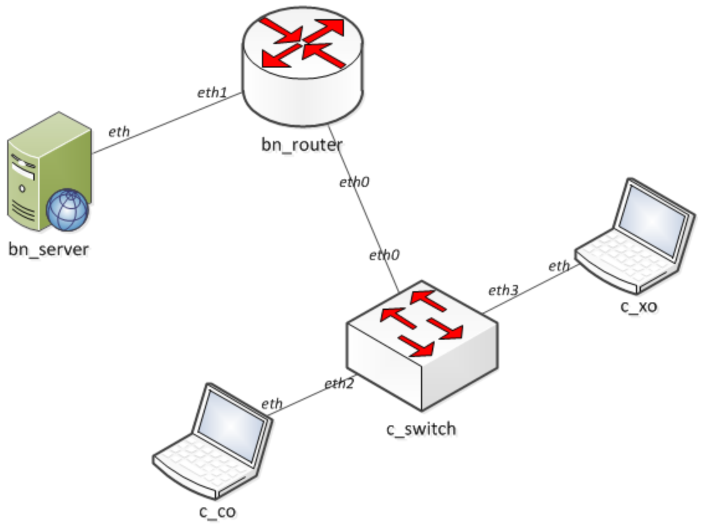

Figure 5.9: Physical topology of example network with single server

5.3.1 Positive Derivations

As with the previous scenarios, we begin by pursuing the positive conclusions. Here, we first derive $D \vdash (HTTPS\_RQ, c\_co\_eth_e) \to (HTTPS\_RQ, bn\_server\_eth_i)$. In plain English, we seek to show that the Battery C commander's host can access the battalion's SharePoint service. Second, we derive $D \vdash (IRC\_PRIVMSG, c\_xo\_eth_e) \to (IRC\_PRIVMSG, bn\_server\_eth_i)$. In plain English, we then seek to show that the Battery C executive officer's host can access the battalion's IRC service.

Our strategy here is to search the network semantics for a rule to help us incrementally derive the final conclusion. We begin by applying the (HTTPS_RQ-ENCAP) rule:

$$\frac{\begin{array}{l} a_1 = HTTPS\_RQ \\ s_1 = (\gamma_1, \gamma_2, \gamma_3, a_1) \\ \gamma_1 = tcp \\ \gamma_2 \in (D\,c\_co).portrange \end{array}}{D \vdash (a_1, c\_co\_eth_e) \to (s_1, c\_co\_eth_e)}$$

The derivation continues in this way, similar to the previous scenarios, until we reach the destination and apply the (HTTPS_RQ-DECAP) rule:

$$\frac{\gamma_3 = 443}{D \vdash (s_1, bn\_server\_eth_i) \to (a_1, bn\_server\_eth_i)}$$

Using the (TRANS) rule, we finally reach the desired conclusion:

$$\frac{\begin{array}{l} D \vdash (a_1, c\_co\_eth_i) \to (s_1, c\_co\_eth_i) \\ D \vdash (s_1, c\_co\_eth_i) \to (a_1, bn\_server\_eth_i) \end{array}}{D \vdash (a_1, c\_co\_eth_i) \to (a_1, bn\_server\_eth_i)}$$

Having completed this derivation, we can prove the first part of our desired positive outcomes. We continue to derive the second desired conclusion, that the Battery C executive officer's host can access the battalion's IRC service. We complete this derivation in a manner very similar to the first derivation. Figure 5.10 shows an excerpt of the results of the derivations.

5.3.2 Negative Derivations

Having completed the positive derivations, we now pursue the negative derivations. However, we cannot derive the first desired conclusion without creating a conflict with a constraint generated in one of the positive derivations. We conduct an exhaustive search for a derivation in the access control logic and find that one does not exist. Described here is one attempt at the link layer of the network stack, one attempt at the network layer, and one attempt at the transport layer.

$$
\begin{array}{l}
a_1 = HTTPS\_RQ \\
s_1 = (\gamma_1, \gamma_2, \gamma_3, a_1) \\
\gamma_1 = tcp \\
\ldots \\
(\gamma_1, \gamma_2) \notin (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[srcport] \\
(\gamma_1, \gamma_2) \notin (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[dstport] \\
\gamma_3 = 443 \\
a_2 = IRC\_PRIVMSG \\
s_2 = (\gamma_{15}, \gamma_{16}, \gamma_{17}, a_2) \\
\gamma_{15} = tcp \\
\ldots \\
(\gamma_{15}, \gamma_{16}) \notin (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[srcport] \\
(\gamma_{15}, \gamma_{16}) \notin (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[dstport] \\
\gamma_{17} = 6667
\end{array}
$$

Figure 5.10: Excerpt of fresh variables generated during positive derivations, third scenario

Link Layer

In this attempt, we decide to focus on MAC filtering at the device *bn_router*. We begin by making an assumption about connectivity between *bn_router* and *bn_server*. We then apply the (LINK-FILTER-DSTHW) and (PKT-ENCAP) rules for instrumentation:

$$
\frac{\gamma_{21} \in (D\,bn\_router).ifaces[bn\_router\_eth0].ifilter[srchw]}{D, \emptyset \vdash (f_4, c\_switch\_eth0_e) \not\rightarrow (f_4, bn\_router\_eth0_i)}
$$

$$
\frac{D, \emptyset \vdash (f_4, c\_switch\_eth0_e) \not\rightarrow (f_4, bn\_router\_eth0_i)}{D, \emptyset \vdash (p_2, c\_switch\_eth0_e) \not\rightarrow (p_2, bn\_router\_eth0_i)}
$$

The assumption is discharged using the (LINK-FWD) rule:

$$
\frac{\begin{array}{c} D, A_1 \vdash (p_2, bn\_router\_eth1_e) \not\rightarrow (p_2, bn\_server\_eth_i) \\ D, \emptyset \vdash (p_2, c\_switch\_eth0_e) \not\rightarrow (p_2, bn\_router\_eth0_i) \end{array}}{D, \emptyset \vdash (p_2, c\_switch\_eth0_e) \not\rightarrow (p_2, bn\_server\_eth_i)}
$$

From here, we use the (TERMINAL) rule to reach back to the source, and then we apply the encapsulation rules (SEG-ENCAP) and (IRC_PRIVMSG-ENCAP) to complete the derivation. The result is shown in Figure 5.11.

$$A_1 = (p_2, bn\_router\_eth1_e) \not\rightarrow (p_2, bn\_server\_eth_i)$$
$$\gamma_{21} \in (D\, bn\_router).ifaces[bn\_router\_eth0].ifilter[srchw]$$

Figure 5.11: First attempt, negative derivation, third scenario

As expected, this derivation generates a constraint that contradicts one generated in the positive derivation. By using unification, we find that our negative derivation yields the following:

$$\begin{aligned}&(D\, c\_xo).ifaces[c\_xo\_eth].hwaddr\\ &\quad \in (D\, bn\_router).ifaces[bn\_router\_eth0].ifilter[srchw]\end{aligned}$$

This contradicts the following constraint generated in the positive derivations:

$$\begin{aligned}&(D\, c\_xo).ifaces[c\_xo\_eth].hwaddr\\ &\quad \notin (D\, bn\_router).ifaces[bn\_router\_eth0].ifilter[srchw]\end{aligned}$$

A similar conflict arises after every attempt to introduce MAC filtering. We must move on to instrumentation at the network layer.

Network Layer

In this attempt, we focus on IP packet filtering as the means of instrumentation, specifically at *bn_router* again. We begin by making an assumption about connectivity between *bn_router* and *bn_server* at the network layer. Our instrumentation choice calls for the (LINK-FILTER-DSTIP) rule:

$$\frac{\gamma_{19} \in (D\, bn\_router).ifaces[bn\_router\_eth0].ifilter[dstip]}{D, \emptyset \vdash (p_2, c\_switch\_eth0_e) \not\rightarrow (p_2, bn\_router\_eth0_i)}$$

We discharge our assumptions with the (LINK-FWD) rule and then complete the derivation as we did in the previous attempt. The results are shown in Figure 5.12.

As expected, a conflict exists in the resulting constraint set. In the negative derivation, we

$$A_1 = (p_2, bn\_router\_eth1_e) \not\rightarrow (p_2, bn\_server\_eth_i)$$
$$\gamma_{19} \in (D\,bn\_router).ifaces[bn\_router\_eth0].ifilter[dstip]$$

Figure 5.12: Second attempt, negative derivation, third scenario

find the following:

$$\begin{aligned}&(D\,bn\_server).ifaces[bn\_server\_eth].ipaddr\\&\quad\in (D\,bn\_router).ifaces[bn\_router\_eth0].ifilter[dstip]\end{aligned}$$

This contradicts the following constraint generated in the positive derivations:

$$\begin{aligned}&(D\,bn\_server).ifaces[bn\_server\_eth].ipaddr\\&\quad\notin (D\,bn\_router).ifaces[bn\_router\_eth0].ifilter[dstip]\end{aligned}$$

We attempt all other derivations using packet filters and forwarding table constraints, and we find similar conflicts each time. We must move from here to the transport layer.

Transport Layer

The logic offers one option for access control at the transport layer (port filtering at the server), so we attempt to use that to complete the derivation. At this level of the network stack, our derivation consists of using two rules, (LINK-FILTER-DSTPORT) and (IRC_PRIVMSG-ENCAP) to reach the final conclusion:

$$\frac{(\gamma_{15}, \gamma_{17}) \in (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[dstport]}{D, \emptyset \vdash (s_2, c\_co\_eth_e) \not\rightarrow (s_2, bn\_server\_eth_i)}$$

$$\frac{D, \emptyset \vdash (s_2, c\_co\_eth_e) \not\rightarrow (s_2, bn\_server\_eth_i)}{D, \emptyset \vdash (a_2, c\_co\_eth_e) \not\rightarrow (a_2, bn\_server\_eth_i)}$$

Still, we find a conflict between this and the positive derivations. Our negative derivation yields the following constraint:

$$(tcp, 6667) \in (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[dstport]$$

This contradicts the following constraint generated in the positive derivations:

$$(tcp, 6667) \notin (D\,bn\_server).ifaces[bn\_server\_eth].ifilter[dstport]$$

We have shown that any instrumentation pursuant to a negative outcome creates a conflict with some positive outcome. Specifically, any attempt to restrict the commander's access to the IRC service also cuts the executive officer's access, which violates the service-level requirements. We therefore fail to generate a satisfiable constraint set for this scenario's specified requirements using the given network semantics and logic. In the first two scenarios, we saw how the approach could be successfully used to provide options for constraining a network configuration. In this scenario, we see how the approach shows no way to generate a satisfiable constraint set.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 6: Conclusion

In Chapter 1, we posed five research questions. These questions have guided our thesis research, and they are discussed throughout this thesis. We summarize our findings by revisiting each of those questions here.

Can a network's logical organization be automatically derived based on its physical topology and service requirements? We found that, given an understanding of network behavior, we could derive the relationships between logical parameters that are required to meet the service requirements. There may be multiple derivations, each having a different instantiation of these logical parameters (e.g. number of subnets and VLAN assignments). Which derivation we pick can influence network performance. Choosing the best derivation requires knowing how different choices affect performance (e.g. locating a filter close to an origin can reduce traffic). While choosing the best derivation is beyond the scope of this thesis, we have shown how one can reason about the obligations of a network's logical organization in order to deny access to services for some users while guaranteeing access to them for others. This is a major step away from the current practice of first creating a logical organization and then *hoping* it meets access control needs.

What constitutes a logical organization? Section 3.2 directly addresses this question. We realize that a network's logical organization is a function of the network's devices and capabilities. For example, if the network includes switches with VLAN capabilities, the logical organization will include VLAN assignments for switch ports. Therefore, the parameters that constitute a logical organization could become quite large. In this work, we attempt to capture the lowest common denominator and provide a basic set of parameters to show the service-oriented access control framework's key concepts.

What constitutes a network service? This question continues to generate discussion. During our work, we considered the information required to describe the service so that the network can properly support it. With that in mind, we can certainly say that a communication protocol is essential to a network service. In fact, we found that the communication protocol sufficiently describes the network service, at least for our purposes.

What constitutes a service specification? We found that we could effectively specify a service with a sequence of rewrite operations representing messages passed between hosts in accordance with a protocol. This sequence of rewrites provides enough information for us to constrain a logical organization. A whitepaper [32] provides an example of this. It specifies HTTP with a regular expression of rewrites representing TCP and HTTP messages sent between a client and a web server. In this work, however, we focus on individual rewrite operations.

How can a network's logical organization change based on the service specification? Simply put, a network's logical organization can change drastically depending on the network's service requirements. As Chapter 5 shows, we had to consider several different options for instrumenting the network – and thereby changing the network's logical organization – in order to find a solution that allowed to us achieve both positive and negative service requirements. This task became more difficult as the number of requirements increased.

This thesis describes a logical system that allows us to systematically reason about denying movement of a message across a network. The system ensures this for all possible paths through the network. The system also provides formal relationships between application-layer messages and underlying protocols, thereby bridging the gap between lower-level device configuration directives and their effects on application-layer messages. It is important to note that the system bridges this gap using standards and codified network behavior as opposed to experience and industry best practice, which makes this work unique compared to other current research efforts, as we discuss in Chapter 2.

The access control logic was applied in Chapter 5 to realistic scenarios that have relevance in the DOD. To achieve this, we use operational experience in building and maintaining networks for USMC field artillery battalions. In these scenarios, we also show how our approach is ideally suited for small-scale tactical networks with fine-grained security and availability requirements.

It is also important to reiterate that the model (semantics) described in Chapter 3, against which the soundness of the access control logic is measured, comes from previous work

[32]. The access control logic, which is the primary contribution of this thesis, takes a step toward the overall goal of automatic configuration of networks based on service-level requirements, both positive and negative. We reproduce the model in this thesis solely for the purpose of keeping the thesis self-contained.

An important issue for discussion in future work is addressing the effect of dynamic events on the network's logical organization. In this thesis, logical organization is assumed not to be a function of dynamic events like link or device failure. In reality, however, these events can certainly affect logical organization and thereby affect access control. To address dynamic changes in physical topology, we envision a system that can detect such a topology-changing event, generate and solve a new SOAC instance, and then apply the resulting logical organization to the network in response to the event.

An algorithm should be developed for computing the best derivation in the access control logic for a given service. The notion of "best" needs attention. Once addressed, it will also guide the algorithm in its search for the best instantiation of logical parameters constrained by virtue of the derivation. Ideally, an algorithmic solution will compute a derivation, generate constraints on the logical organization based on that derivation, try to satisfy those constraints, and use a satisfying substitution to configure the network's devices. A prototype algorithm has been written in Prolog, but it makes no attempt to find the best instantiation of constraints. Therefore, we further suggest developing heuristics for guiding such an algorithm (e.g., finding the substitution that minimizes traffic flow throughout the network or reduces the sizes of forwarding tables). By doing so, a network administrator needs only to concern himself with service requirements, letting the SOAC framework optimize the network's performance and ensure its security. Therefore, network complexity can be hidden from administrators who need only specify services and their authorized users, not the low-level device-specific commands that may collectively achieve this effect.

The network semantics and access control logic might be extended to capture more device capabilities and underlying network protocols. In this work, we focus on a small set of devices with a small set of capabilities, and we investigate a small set of services. Future work should enlarge these sets so that the SOAC approach can leverage more advanced but commonly used technologies like VLAN, virtual private network (VPN), and NAT. Furthermore, future work in this area should focus on prioritizing service requirements. When supporting multiple services, network administrators must often make tradeoffs between the security and functionality of those various services. This is especially true in tactical networks, where bandwidth may be limited and security of certain services may be critical. We therefore suggest future work in capturing this notion of priority in leveraging QoS technologies to build a priority-sensitive SOAC approach.

APPENDIX A:
Network Semantics

$$\text{(TRANS)}\quad \frac{\begin{array}{l} D \vdash (x,u) \to (y,v) \\ D \vdash (y,v) \to (z,w) \end{array}}{D \vdash (x,u) \to (z,w)}$$

$$\text{(FRAME-DECAP)}\quad \frac{\begin{array}{l} switch \notin (D\,n).type \\ p \in P \\ f \in F \\ f.data = p \\ w \in (D\,n).ifaces \\ p.srcip \notin (D\,n).ifaces[w].ifilter[srcip] \\ p.dstip \notin (D\,n).ifaces[w].ifilter[dstip] \end{array}}{D \vdash (f,w_i) \to (p,w_i)}$$

$$\text{(FRAME-ENCAP)}\quad \frac{\begin{array}{l} switch \notin (D\,n).type \\ p \in P \\ f \in F \\ f.data = p \\ w \in (D\,n).ifaces \\ f.srchw = (D\,n).ifaces[z].hwaddr \\ f.srchw \notin (D\,n).ifaces[z].efilter[srchw] \\ f.dsthw \notin (D\,n).ifaces[z].efilter[dsthw] \end{array}}{D \vdash (p,z_e) \to (f,z_e)}$$

$$
(\textsc{switch}) \quad \frac{\begin{array}{l} \mathit{switch} \in (D\,n).\mathit{type} \\ f \in F \\ x, z \in (D\,n).\mathit{ifaces} \\ x \neq z \\ f.\mathit{srchw} \notin (D\,n).\mathit{ifaces}[z].\mathit{efilter}[\mathit{srchw}] \\ f.\mathit{dsthw} \notin (D\,n).\mathit{ifaces}[z].\mathit{efilter}[\mathit{dsthw}] \end{array}}{D \vdash (f, x_i) \rightarrow (f, z_e)}
$$

$$
(\textsc{frame-tx-switch-rx}) \quad \frac{\begin{array}{l} \mathit{switch} \in (D\,n).\mathit{type} \\ f \in F \\ y \in (D\,m).\mathit{ifaces}[w].\mathit{direct} \\ m \neq n \\ f.\mathit{srchw} \notin (D\,n).\mathit{ifaces}[y].\mathit{ifilter}[\mathit{srchw}] \\ f.\mathit{dsthw} \notin (D\,n).\mathit{ifaces}[y].\mathit{ifilter}[\mathit{dsthw}] \end{array}}{D \vdash (f, w_e) \rightarrow (f, y_i)}
$$

$$
(\textsc{frame-tx-rx}) \quad \frac{\begin{array}{l} \mathit{switch} \notin (D\,n).\mathit{type} \\ f \in F \\ y \in (D\,m).\mathit{ifaces}[w].\mathit{direct} \\ m \neq n \\ f.\mathit{dsthw} = (D\,n).\mathit{ifaces}[y].\mathit{hwaddr} \\ f.\mathit{srchw} \notin (D\,n).\mathit{ifaces}[y].\mathit{ifilter}[\mathit{srchw}] \\ f.\mathit{dsthw} \notin (D\,n).\mathit{ifaces}[y].\mathit{ifilter}[\mathit{dsthw}] \end{array}}{D \vdash (f, w_e) \rightarrow (f, y_i)}
$$

$$(\text{PACKET-DECAP})\quad \frac{\begin{array}{l} switch \notin (Dn).type \\ router \notin (Dn).type \\ s \in S \\ p \in P \\ p.data = s \\ w \in (Dn).ifaces \\ p.dstip = (Dn).ifaces[w].ipaddr \\ (s.protocol, s.srcport) \notin (Dn).ifaces[w].ifilter[srcport] \\ (s.protocol, s.dstport) \notin (Dn).ifaces[w].ifilter[dstport] \end{array}}{D \vdash (p, w_i) \rightarrow (s, w_i)}$$

$$(\text{PACKET-ENCAP})\quad \frac{\begin{array}{l} switch \notin (Dn).type \\ router \notin (Dn).type \\ s \in S \\ p \in P \\ p.data = s \\ w \in (Dn).ifaces \\ (Dn).ifaces[w].netmask = m \\ p.dstip \ \& \ m = (Dn).ifaces[w].dest \\ p.srcip = (Dn).ifaces[w].ipaddr \\ p.srcip \notin (Dn).ifaces[w].efilter[srcip] \\ p.dstip \notin (Dn).ifaces[w].efilter[dstip] \end{array}}{D \vdash (s, w_e) \rightarrow (p, w_e)}$$

$$(\textsc{Router})\quad \frac{\begin{array}{l} router \in (Dn).type \\ v, w \in (Dn).ifaces \\ v \neq w \\ (Dn).ifaces[w].netmask = mask \\ p \in P \\ p.dstip \neq 255.255.255.255 \\ p.dstip \neq \neg mask \mid (Dn).ifaces[w].dest \\ p.dstip \; \& \; mask = (Dn).ifaces[w].dest \\ (Dn).ifaces[v].dest \neq (Dn).ifaces[w].dest \\ p.srcip \notin (Dn).ifaces[w].efilter[srcip] \\ p.dstip \notin (Dn).ifaces[w].efilter[dstip] \end{array}}{D \vdash (p, v_i) \rightarrow (p, w_e)}$$

$$(\textsc{HTTPS\_RQ-Decap})\quad \frac{\begin{array}{l} server \in (Dn).type \\ s \in S \\ a = HTTPS\_RQ \\ s.data = a \\ w \in (Dn).ifaces \\ s.dstport = 443 \end{array}}{D \vdash (s, w_i) \rightarrow (a, w_i)}$$

$$(\textsc{HTTPS\_RQ-Encap})\quad \frac{\begin{array}{l} host \in (Dn).type \\ s \in S \\ a = HTTPS\_RQ \\ s.data = a \\ w \in (Dn).ifaces \\ s.protocol = tcp \\ s.srcport \in (Dn).portrange \end{array}}{D \vdash (a, w_e) \rightarrow (s, w_e)}$$

$$(\textsc{IRC\_PRIVMSG-DECAP})\qquad \frac{\begin{array}{l} server \in (D\,n).type \\ s \in S \\ a = IRC\_PRIVMSG \\ s.data = a \\ w \in (D\,n).ifaces \\ s.dstport = 6667 \end{array}}{D \vdash (s, w_i) \rightarrow (a, w_i)}$$

$$(\textsc{IRC\_PRIVMSG-ENCAP})\qquad \frac{\begin{array}{l} host \in (D\,n).type \\ s \in S \\ a = IRC\_PRIVMSG \\ s.data = a \\ w \in (D\,n).ifaces \\ s.protocol = tcp \\ s.srcport \in (D\,n).portrange \end{array}}{D \vdash (a, w_e) \rightarrow (s, w_e)}$$

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B: Access Control Logic

$$\text{(LINK-FILTER-SRCHW)}\quad \frac{\begin{array}{l} host \notin (D\,n).type \\ f \in F \\ u \in (D\,m).ifaces \\ f.srchw \in (D\,n).ifaces[v].ifilter[srchw] \\ m \neq n \end{array}}{D,A \vdash (f,u_e) \not\rightarrow (f,v_i)}$$

$$\text{(LINK-FILTER-SRCIP)}\quad \frac{\begin{array}{l} switch \notin (D\,n).type \\ host \notin (D\,n).type \\ p \in P \\ u \in (D\,m).ifaces \\ p.srcip \in (D\,n).ifaces[v].ifilter[srcip] \\ m \neq n \end{array}}{D,A \vdash (p,u_e) \not\rightarrow (p,v_i)}$$

$$\text{(LINK-FILTER-DSTHW)}\quad \frac{\begin{array}{l} host \notin (D\,n).type \\ f \in F \\ u \in (D\,m).ifaces \\ f.dsthw \in (D\,n).ifaces[v].ifilter[dsthw] \\ m \neq n \end{array}}{D,A \vdash (f,u_e) \not\rightarrow (f,v_i)}$$

$$\textsc{(Link-Filter-Dstip)}\quad \frac{\begin{array}{l} \mathit{switch} \notin (D\,n).\mathit{type} \\ \mathit{host} \notin (D\,n).\mathit{type} \\ p \in P \\ u \in (D\,m).\mathit{ifaces} \\ p.\mathit{dstip} \in (D\,n).\mathit{ifaces}[v].\mathit{ifilter}[\mathit{dstip}] \\ m \neq n \end{array}}{D, A \vdash (p, u_e) \nrightarrow (p, v_i)}$$

$$\textsc{(Link-Filter-Dstport)}\quad \frac{\begin{array}{l} \mathit{server} \in (D\,n).\mathit{type} \\ s \in S \\ u \in (D\,m).\mathit{ifaces} \\ (s.\mathit{protocol}, s.\mathit{dstport}) \in (D\,n).\mathit{ifaces}[v].\mathit{ifilter}[\mathit{dstport}] \\ m \neq n \end{array}}{D, A \vdash (s, u_e) \nrightarrow (s, v_i)}$$

$$\textsc{(Device-Filter-Srchw)}\quad \frac{\begin{array}{l} \mathit{server} \notin (D\,n).\mathit{type} \\ \mathit{host} \notin (D\,n).\mathit{type} \\ f \in F \\ u, v \in (D\,m).\mathit{ifaces} \\ f.\mathit{srchw} \in (D\,m).\mathit{ifaces}[v].\mathit{efilter}[\mathit{srchw}] \end{array}}{D, A \vdash (f, u_i) \nrightarrow (f, v_e)}$$

$$\textsc{(Device-Filter-Srcip)}\quad \frac{\begin{array}{l} \mathit{router} \in (D\,n).\mathit{type} \\ p \in P \\ u, v \in (D\,m).\mathit{ifaces} \\ p.\mathit{srcip} \in (D\,m).\mathit{ifaces}[v].\mathit{efilter}[\mathit{srcip}] \end{array}}{D, A \vdash (p, u_i) \nrightarrow (p, v_e)}$$

$$\textsc{(device-filter-dsthw)}\quad \frac{\begin{array}{l} server \notin (D\,n).type \\ host \notin (D\,n).type \\ f \in F \\ u,v \in (D\,m).ifaces \\ f.dsthw \in (D\,m).ifaces[v].efilter[dsthw] \end{array}}{D,A \vdash (f,u_i) \not\rightarrow (f,v_e)}$$

$$\textsc{(device-filter-dstip)}\quad \frac{\begin{array}{l} router \in (D\,n).type \\ p \in P \\ u,v \in (D\,m).ifaces \\ p.dstip \in (D\,m).ifaces[v].efilter[dstip] \end{array}}{D,A \vdash (p,u_i) \not\rightarrow (p,v_e)}$$

$$\textsc{(terminal)}\quad \frac{\begin{array}{l} (D\,m).ifaces[u].direct = \{v\} \\ (D\,n).ifaces = \{v\} \\ v \neq x \end{array}}{D,A \vdash (d,u_e) \not\rightarrow (d,x_i)}$$

$$\textsc{(ip-address)}\quad \frac{\begin{array}{l} p \in P \\ u \in (D\,m).ifaces \\ p.dstip \neq (D\,n).ifaces[v].ipaddr \\ p.dstip \neq 255.255.255.255 \\ router \notin (D\,n).type \\ switch \notin (D\,n).type \end{array}}{D,A \vdash (p,u_e) \not\rightarrow (p,v_i)}$$

$$\text{(SUBNET)}\quad \frac{\begin{array}{l} u,v \in (D\,n).ifaces \\ p \in P \\ (D\,n).ifaces[v].netmask = mask \\ p.dstip \mathbin{\&} mask \neq (D\,n).ifaces[v].dest \\ router \in (D\,n).type \end{array}}{D,A \vdash (p,u_i) \not\rightarrow (p,v_e)}$$

$$\text{(LINK-ASSUMPTION)}\quad \frac{(d,u_e) \not\rightarrow (d,v_i) \in A}{D,A \vdash (d,u_e) \not\rightarrow (d,v_i)}$$

$$\text{(DEVICE-ASSUMPTION)}\quad \frac{(d,u_i) \not\rightarrow (d,v_e) \in A}{D,A \vdash (d,u_i) \not\rightarrow (d,v_e)}$$

$$\text{(FWD)}\quad \frac{\begin{array}{l} v \in (D\,n).ifaces \\ (D\,m).ifaces[u].direct = \{v\} \\ \forall w \in (D\,n).ifaces - \{v\}.\, D,A \vdash (d,w_e) \not\rightarrow (d,x_i) \end{array}}{D,A \vdash (d,u_e) \not\rightarrow (d,x_i)}$$

$$\text{(LINK-FWD)}\quad \frac{\begin{array}{l} v \in (D\,n).ifaces \\ (D\,m).ifaces[u].direct = \{v\} \\ \forall w \in (D\,n).ifaces - \{v\}.\, D,A \cup S \vdash (d,w_e) \not\rightarrow (d,x_i) \\ D,A \vdash (d,u_e) \not\rightarrow (d,v_i) \end{array}}{D,A \vdash (d,u_e) \not\rightarrow (d,x_i)}$$

$$\text{(DEVICE-FWD)}\quad \frac{\begin{array}{l} v \in (D\,n).ifaces \\ (D\,m).ifaces[u].direct = \{v\} \\ \forall w \in (D\,n).ifaces - \{v\}.\, D,A \cup S \vdash (d,w_e) \not\rightarrow (d,x_i) \\ \forall w \in (D\,n).ifaces - \{v\}.\, D,A \vdash (d,v_i) \not\rightarrow (d,w_e) \end{array}}{D,A \vdash (d,u_e) \not\rightarrow (d,x_i)}$$

$$(\textsc{HTTPS\_RQ-ENCAP})\quad \frac{\begin{array}{l} s \in S \\ s.data = \alpha \\ \alpha = HTTPS\_RQ \\ D,A \vdash (s,u_e) \not\rightarrow (s,v_i) \end{array}}{D,A \vdash (\alpha,u_e) \not\rightarrow (\alpha,v_i)}$$

$$(\textsc{IRC\_PRIVMSG-ENCAP})\quad \frac{\begin{array}{l} s \in S \\ s.data = \alpha \\ \alpha = IRC\_PRIVMSG \\ D,A \vdash (s,u_e) \not\rightarrow (s,v_i) \end{array}}{D,A \vdash (\alpha,u_e) \not\rightarrow (\alpha,v_i)}$$

$$(\textsc{SEG-ENCAP})\quad \frac{\begin{array}{l} p \in P \\ p.data = s \\ D,A \vdash (p,u_e) \not\rightarrow (p,v_i) \end{array}}{D,A \vdash (s,u_e) \not\rightarrow (s,v_i)}$$

$$(\textsc{PKT-ENCAP})\quad \frac{\begin{array}{l} f \in F \\ f.data = p \\ D,A \vdash (f,u_e) \not\rightarrow (f,v_i) \end{array}}{D,A \vdash (p,u_e) \not\rightarrow (p,v_i)}$$

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] M. Riley *et al.* (2014, Mar.). Missed alarms and 40 million stolen credit card numbers: How Target blew it. *Bloomberg Businessweek.* [Online]. Available: http://www.businessweek.com/articles/2014-03-13/target-missed-alarms-in-epic-hack-of-credit-card-data

[2] J. Vijayan. (2007, Mar.). TJX data breach: At 45.6M card numbers, it's the biggest ever. *Computerworld.* [Online]. Available: http://www.computerworld.com/article/2544306/security0/tjx-data-breach--at-45-6m-card-numbers--it-s-the-biggest-ever.html

[3] S. Narain *et al.*, "Declarative infrastructure configuration synthesis and debugging," *J. Netw. Syst. Manage.*, vol. 16, no. 3, pp. 235–258, Oct. 2008.

[4] T. L. Hinrichs *et al.*, "Practical declarative network management," in *Proc. 1st ACM Workshop on Research on Enterprise Networking*, Barcelona, Spain, 2009, pp. 1–10.

[5] N. Gude *et al.*, "NOX: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[6] Open Networking Foundation. (2014, May). OpenFlow. [Online]. Available: http://www.opennetworking.org

[7] X. Chen *et al.*, "Declarative configuration management for complex and dynamic networks," in *Proc. 6th ACM Int. Conf. on Emerging Networking Experiments and Technologies*, Philadelphia, PA, 2010, pp. 6:1–6:12.

[8] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[9] R. Soulé *et al.*, "Managing the network with Merlin," in *Proc. 12th ACM Workshop on Hot Topics in Networks*, College Park, MD, 2013, pp. 24:1–24:7.

[10] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Proc. 13th Int. Symp. on Practical Aspects of Declarative Languages*, Austin, TX, 2011, pp. 235–249.

[11] N. Foster *et al.*, "Frenetic: A network programming language," *SIGPLAN Not.*, vol. 46, no. 9, pp. 279–291, Sep. 2011.

[12] Y. E. Sung *et al.*, "Towards systematic design of enterprise networks," in *Proc. 4th ACM Int. Conf. on Emerging Networking Experiments and Technologies*, Madrid, Spain, 2008, pp. 22:1–22:12.

[13] G. G. Xie *et al.*, “On static reachability analysis of IP networks,” in *Proc. 24th Annu. Joint Conf. of IEEE Computer and Communications Societies*, vol. 3, Miami, FL, 2005, pp. 2170–2183.

[14] A. Williams. (2002, Sep.). Requirements for automatic configuration of IP hosts. [Online]. Available: http://www.ietf.org/archive/id/draft-ietf-zeroconf-reqts-12.txt

[15] E. Guttman. (2001, Jul.). Zeroconf host profile. [Online]. Available: http://www.ietf.org/archive/id/draft-ietf-zeroconf-host-prof-01.txt

[16] Apple. Apple Bonjour. [Online]. Available: http://developer.apple.com/bonjour

[17] *Dynamic Configuration of IPv4 Link-Local Addresses*, IETF RFC 3927, May 2005.

[18] *Multicast DNS*, IETF RFC 6762, Feb. 2013.

[19] *DNS-Based Service Discovery*, IETF RFC 6763, Feb. 2013.

[20] C. Akinlar *et al.*, “An IP address configuration algorithm for multi-router Zeroconf networks,” in *Proc. 7th IEEE Int. Symp. on Computers and Communications*, Taormina, Italy, 2002, pp. 462–467.

[21] C. Akinlar and A. Shankar, “IPv4 auto-configuration of multi-router Zeroconf networks with unique subnets,” in *Proc. 4th Int. Conf. on Networking*, Reunion Island, France, 2005, pp. 156–163.

[22] N. Feamster and H. Balakrishnan, “Detecting BGP configuration faults with static analysis,” in *Proc. 2nd USENIX Symp. on Networked Systems Design and Implementation*, Boston, MA, 2005, pp. 43–56.

[23] H. Mai *et al.*, “Debugging the data plane with Anteater,” in *Proc. ACM SIGCOMM 2011 Conf.*, Toronto, Ontario, Canada, 2011, pp. 290–301.

[24] P. Kazemian *et al.*, “Header space analysis: Static checking for networks,” in *Proc. 9th USENIX Symp. on Networked Systems Design and Implementation*, San Jose, CA, 2012, pp. 113–126.

[25] *Requirements for IP Version 4 Routers*, IETF RFC 1812, Jun. 1995.

[26] *Classless Inter-Domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*, IETF RFC 4632, Aug. 2006.

[27] *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, IEEE Std. 802.3-2005, Dec. 2008.

[28] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11-2012, Mar. 2012.

[29] *Internet Protocol*, IETF RFC 791, Sep. 1981.

[30] *Transmission Control Protocol*, IETF RFC 793, Sep. 1981.

[31] *User Datagram Protocol*, IETF RFC 768, Aug. 1980.

[32] D. Volpano, "Service-oriented automatic configuration," unpublished.

[33] A. V. Aho *et al.*, *Compilers: Principles, Techniques, and Tools*, 2nd ed. Boston, MA: Addison-Wesley, 2006.

[34] D. Volpano, private communication, May 2014.

[35] G. Tourlakis, *Mathematical Logic*. Hoboken, NJ: Wiley, 2008.

[36] *Tactics, Techniques, and Procedures for Field Artillery Manual Cannon Gunnery*, FM 6-40, United States Army, Washington, DC, Apr. 1996.

[37] *Tactics, Techniques, and Procedures for the Field Artillery Cannon Battery*, FM 6-50, United States Army, Washington, DC, Apr. 1996.

[38] K. McMullen, private communication, Apr. 2014.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California